

Tektronix®

8560

**MULTI-USER SOFTWARE
DEVELOPMENT UNIT**

DIGITAL DESIGN LAB

SYSTEM USERS MANUAL



This manual supports the following TEKTRONIX products:

8550 Option	Products	8540 Option	Products
2H	8300E10	2H	8300E10 Option 01
3K	8300P10	3K	8300P10
3L	8300P11	3L	8300P11
3M	8300P12	3M	8300P12
3N	8300P13	3N	8300P13

This manual supports a software/firmware module that is compatible with:

DOS/50 Version 2 (8550)
OS/40 Version 1 (8540)

**PLEASE CHECK FOR CHANGE INFORMATION
AT THE REAR OF THIS MANUAL.**

8500
MODULAR MDL SERIES
8048/8021/8041A/8022
EMULATOR SPECIFICS
USERS MANUAL

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

070-3967-01
Product Group 61

Serial Number _____

First Printing APR 1982
Revised SEP 1983

LIMITED RIGHTS LEGEND

Software License No. _____

Contractor: Tektronix, Inc.

Explanation of Limited Rights Data Identification Method

Used: Entire document subject to limited rights.

Those portions of this technical data indicated as limited rights data shall not, without the written permission of the above Tektronix, be either (a) used, released or disclosed in whole or in part outside the Customer, (b) used in whole or in part by the Customer for manufacture or, in the case of computer software documentation, for preparing the same or similar computer software, or (c) used by a party other than the Customer, except for: (i) emergency repair or overhaul work only, by or for the Customer, where the item or process concerned is not otherwise reasonably available to enable timely performance of the work, provided that the release or disclosure hereof outside the Customer shall be made subject to a prohibition against further use, release or disclosure; or (ii) release to a foreign government, as the interest of the United States may require, only for information or evaluation within such government or for emergency repair or overhaul work by or for such government under the conditions of (i) above. This legend, together with the indications of the portions of this data which are subject to such limitations shall be included on any reproduction hereof which includes any part of the portions subject to such limitations.

RESTRICTED RIGHTS IN SOFTWARE

The software described in this document is licensed software and subject to **restricted rights**. The software may be used with the computer for which or with which it was acquired. The software may be used with a backup computer if the computer for which or with which it was acquired is inoperative. The software may be copied for archive or backup purposes. The software may be modified or combined with other software, subject to the provision that those portions of the derivative software incorporating restricted rights software are subject to the same restricted rights.

Copyright © 1982 Tektronix, Inc. All rights reserved. Contents of this publication may not be reproduced in any form without the written permission of Tektronix, Inc.

Products of Tektronix, Inc. and its subsidiaries are covered by U.S. and foreign patents and/or pending patents.

TEKTRONIX, TEK, SCOPE-MOBILE, and  are registered trademarks of Tektronix, Inc. TELEQUIPMENT is a registered trademark of Tektronix U.K. Limited.

Printed in U.S.A. Specification and price change privileges are reserved.

Section 7H

8048/8021/8041A/8022 EMULATOR SPECIFICS

	Page
<u>Introduction</u>	7H-1
<u>General Information</u>	7H-1
8048/8021/8041A/8022 Emulator Hardware Configurations	7H-1
Terminology	7H-2
<u>Emulator-Specific Commands, Parameters, and Displays</u>	7H-2
SEL --- Selecting an Emulator	7H-2
Byte/Word Parameter	7H-2
Symbolic Debug	7H-2
MAP --- Mapping Memory	7H-3
EVT Address Parameter	7H-3
TTA Bus Operation Designators	7H-3
Setting Breakpoints	7H-4
Memory Allocation Commands	7H-4
Register Designators	7H-4
DS --- Sample Status Display	7H-6
RESET --- Resetting Emulator Status	7H-7
DI --- Sample Disassembled Code	7H-8
TRA --- Sample TRAcE Display	7H-8
<u>Service Calls</u>	7H-10
SVCs in Modes 1 and 2	7H-10
SVC Demonstration	7H-11
SVC Design Features	7H-14
<u>Special Considerations</u>	7H-16
Clock Rate	7H-16
Memory Mapping	7H-17
Emulation Modes	7H-21
NOP--MOVX Instruction Sequence	7H-21
TRA and the Timer/Counter	7H-21
8048/8049 Special Considerations	7H-22
8021 Special Considerations	7H-23
8041A Special Considerations	7H-23
8022 Special Considerations	7H-24
<u>Emulator Timing</u>	7H-25
Probe/Prototype Interface Diagrams	7H-31
<u>Installing Your 8048/8021/8041A/8022 Emulator Software</u>	7H-34
8540 Firmware Installation Procedure	7H-34
8550 Installation Procedure	7H-34
<u>8048/8021/8041A/8022 Demonstration Run</u>	7H-36
Introduction	7H-36
Examine the Demonstration Run Program	7H-39
Assemble and Load the Demonstration Program	7H-40
Case 1: Assemble and Load on the 8550	7H-41
Case 2: Assemble on the 8560; Download to the 8540	7H-47

Case 3: Download from Your Host to the 8540	7H-53
Case 4: Patch the Program into Memory	7H-56
Run the Demonstration Program	7H-58
Monitor Program Execution	7H-61
Summary of 8048/8021/8041A/8022 Emulator Demonstration Run	7H-66
Delete the Demonstration Run Files	7H-67

TABLES

Table

No.	
7H-1	Microcomputers Supported by the 8048/8021/8041A/8022 Emulator 7H-1
7H-2	8048/8021/8041A/8022 Bus Operation Designators 7H-3
7H-3	8048/8021/8041A/8022 Registers and Flags 7H-5
7H-4	8048/8021/8041A/8022 Service Calls 7H-10
7H-5	Maximum Allowable Clock Frequencies 7H-16
7H-6	Probe/Prototype Interface Delays for the 8048/8021 7H-25
7H-7	Probe/Prototype Interface Delays for the 8041A 7H-27
7H-8	Representative 8041A Probe/Microcomputer Timing Differences . 7H-27
7H-9	Probe/Prototype Interface Delays for the 8022 7H-29
7H-10	Basic 8560 Editing Commands 7H-49

ILLUSTRATIONS

Fig.

No.	
7H-1	8048/8021/8041A/8022 program status word 7H-6
7H-2	8048 SVC demonstration program listing 7H-12
7H-3	8048/8021/8041A/8022 memory layout 7H-18
7H-4	8048/8021 timing diagrams 7H-26
7H-5	8041A timing diagrams 7H-28
7H-6	8022 timing diagrams 7H-30
7H-7	Block diagram of 8048/8021 probe/prototype interface 7H-31
7H-8	Block diagram of 8041A probe/prototype interface 7H-32
7H-9	Block diagram of 8022 probe/prototype interface 7H-33
7H-10	Demonstration program 7H-37
7H-11	Demonstration program: Extended Tekhex format 7H-38
7H-12	Host computer commands for preparing demonstration program . 7H-54

Section 7H

8048/8021/8041A/8022 EMULATOR SPECIFICS

INTRODUCTION

This section is designed to be inserted into Section 7 of the 8550 System Users Manual (DOS/50 Version 2) or the 8540 System Users Manual. This Emulator Specifics section explains the features of the 8550 and 8540 systems that are unique to the 8048/8021/8041A/8022 emulator. Throughout the section, "your System Users Manual" refers to the 8550 System Users Manual or the 8540 System Users Manual, and "the operating system" refers to DOS/50 Version 2 or OS/40. The 8048/8021/8041A/8022 Demonstration Run is designed to be used with Section 1, the Learning Guide of your System Users Manual; the rest of this section contains reference material.

GENERAL INFORMATION

8048/8021/8041A/8022 EMULATOR HARDWARE CONFIGURATIONS

In order for the 8048/8021/8041A/8022 emulator to function correctly, it must be connected to the prototype control probe that is appropriate for the microcomputer being emulated. Table 7H-1 lists the microcomputers supported by the 8048/8021/8041A/8022 emulator and gives the corresponding hardware configurations.

Table 7H-1
Microcomputers Supported by the 8048/8021/8041A/8022 Emulator

Microcomputer	Hardware Configuration
8041A/8741A	8041A Prototype Control Probe
8022	8022 Prototype Control Probe
8021	8048/8021 Prototype Control Probe with 8021 Prototype Control Probe Adapter (*a)
8048/8648/8748/8035	8048/8021 Prototype Control Probe (*a)
8049/8039/8039-6	8048/8021 Prototype Control Probe (*a)

(*a) DIP switches on the 8048/8021 Prototype Control Probe select the microcomputer to be emulated, the prototype clock rate, and the presence or absence of external program memory. These switches are described in the 8048/8021/8041A/8022 Emulator Processor Installation Manual.

TERMINOLOGY

The term 8048 refers collectively to the 8048 and to those microcomputers that differ from the 8048 only in the form of on-board ROM (the 8648, 8748, and 8035). The term 8048/8049 refers collectively to those microcomputers that support the 8048 instruction set (the 8048, 8648, 8748, 8035, 8049, 8039, and 8039-6). The terms 8048 family and 8048/8021/8041A/8022 refer collectively to all microcomputers supported by the 8048/8021/8041A/8022 emulator, as listed in Table 7H-1. The term 8041A refers to the 8741A as well as to the 8041A.

In this Emulator Specifics section, the term program memory refers to the on-board ROM and/or external memory in which 8048/8021/8041A/8022 program instructions are stored. The term 8550/8540 program memory refers to the RAM in your 8550 or 8540 that may be used as a substitute for memory in the microcomputer and/or prototype. Throughout the other sections of your System Users Manual, the term program memory refers to 8550 or 8540 program memory.

EMULATOR-SPECIFIC COMMANDS, PARAMETERS, AND DISPLAYSSEL --- Selecting an Emulator

The SEL (SElect) command allows you to select the emulator you want to use with your 8550 or 8540. The following command line selects the 8048/8021/8041A/8022 emulator and assembler:

```
> SEL 8048 <CR>
```

Circuitry in the prototype control probe tells the operating system which microcomputer in the 8048 family is being emulated. For more information, refer to the "8048/8021/8041A/8022 Emulator Configurations" discussion at the beginning of this section.

Enter the following command line to select the 8048 assembler on the 8560:

```
$ uP=8048; export uP <CR>
```

Byte/Word Parameter

Several commands offer you the choice of operating on memory on a byte-oriented or word-oriented basis. In affected commands, this choice is represented by the -B or -W parameter. For the 8048/8021/8041A/8022 emulator, the default value is -B (Byte).

Symbolic Debug

The 8048/8021/8041A/8022 emulator supports the use of symbolic debug. Some of the displays in this document include symbolic debug information.

MAP --- Mapping Memory

The MAP command assigns 128-byte blocks of memory either to 8550/8540 program memory or to prototype memory. For the 8048/8021/8041A/8022 emulator, only blocks in the following address ranges can be mapped:

0000--0FFF (8048/8049 program memory)

2000--20FF (8048/8049 external data memory)

NOTE

No memory mapping is possible for the 8021, 8041A, or 8022: these microcomputers do not support external memory.

For more information on memory considerations, refer to the "Special Considerations" discussion later in this section.

EVT Address Parameter

If you are using the Real-Time Prototype Analyzer (RTPA) option with your 8550, the addresses used in the EVT command line must reflect the memory mapping conventions defined in the "Special Considerations" subsection later in this section. (The RTPA option is not supported on the 8540.)

TTA Bus Operation Designators

Table 7H-2 lists the 8048/8021/8041A/8022 bus operation designators recognized by the Trigger Trace Analyzer's BUS command.

Table 7H-2
8048/8021/8041A/8022 Bus Operation Designators

Symbol	Bus Operation Type
CLR	All types
F	Instruction fetches
I	I/O operations (SVCs only)
NF	Non-fetches
M	Memory accesses
RD	Reads
WT	Writes

Setting Breakpoints

The 8048/8021/8041A/8022 emulator allows you to specify up to two breakpoints with the BK (Breakpoint) command. Breakpoints should be restricted to the ranges 0--OFFF (program memory) and 2000--20FF (8048/8049 external data memory). The 8048/8021/8041A/8022 emulator cannot monitor accesses to internal data memory. Refer to the "Special Considerations" discussion later in this section for information on the memory limitations of your microcomputer.

Memory Allocation Commands

The Memory Allocation Controller (MAC) option cannot be used with the 8048/8021/8041A/8022 emulator. The 8048/8021/8041A/8022 emulator does not use the MEMSP command, and does not support memory space qualifiers or expressions. The 8048 family emulator supports the AL (ALlocate) command, as described in the Command Dictionary of your System Users Manual. The DEAL, MEM, and NOMEM commands are not valid with the 8048/8021/8041A/8022 emulator.

NOTE

You should avoid allocating memory in the range 2000--20FFH and in the range 4000--407FH. These memory ranges are used to emulate External Data Memory and Internal Data Memory, respectively. Allocation of memory in these ranges may yield unexpected results.

Register Designators

Table 7H-3 alphabetically lists the symbols used by DOS/50 and OS/40 to designate the registers and flags used by the microcomputers in the 8048 family. The table provides the following information for each symbol:

- the microcomputers for which that symbol is used
- a description of the register or flag that the symbol represents
- the size of the register or flag
- the value assigned to the register or flag by the RESET command
- whether the register or flag can be assigned a value by the S (Set) command

Figure 7H-1 shows the contents of the 8048/8021/8041A/8022 program status word.

Table 7H-3
8048/8021/8041A/8022 Registers and Flags

DOS/50 or OS/40 Symbol	8 8 8 8 0 0 0 0 4 2 4 2 8 1 1 2	Description	Size in Bits (*a)	Value After RESET (*b)	Altered by S Command? Command?
A	x x x x	Accumulator	8	NC	yes
A0--A7	x x	Alternate registers 0--7	8 each	NC	yes
AC	x x x x	Auxiliary carry flag: bit 6 of PSW	1	0	yes
AN	x	Analog input pin	1	NC	yes
CHIP	x x x x	Microcomputer name	NA	NC	no
CY	x x x x	Carry flag: bit 7 of PSW	1	0	yes
DMA	x	EN DMA instruction exe- cuted since last RESET? 1=yes; 0=no	1	0	no
EI	x x x	External interrupt flag: 1=enabled; 0=disabled	1	0	yes
EPM	x	Switch set to enable external program memory? Y=yes; N=no	NA	NC	no
F0	x x	Flag 0: bit 5 of PSW	1	0	yes
F1	x x	Flag 1	1	0	yes
FLG	x	EN FLAGS instruction exe- cuted since last RESET? 1=yes; 0=no	1	NC	no
IBF	x	Input buffer full flag	1	NC	no
IIP	x x x	Interrupt in progress flag: Y=yes; N=no	NA	N	no
MB	x	Memory bank	1	0	yes
OBF	x	Output buffer full flag	1	NC	no
PC	x x x x	Program counter	12	0	no
PSW	x x x x	Program status word (*c)	8	0 or 8	yes
R0--R7	x x x x	Registers 0--7	8 each	NC	yes
RB	x x	Register bank: bit 4 of PSW	1	0	yes
RETURN	x x x x	Value of last word pushed	16	(*d)	no (*d)
STACK	x x x x	Stack pointer: 08 to 16, derived from bits 0--2 of PSW	NA	08	no (*d)
STF	x	STS flags	4	0	no
TC	x x x x	Timer/counter selector: 0=stop; 1=timer; 2=counter	2	0	yes
TF	x x x x	Timer overflow flag	1	0	yes
TI	x x x	Timer/counter interrupt flag: 1=enabled; 0=disabled	1	0	yes
TR	x x x x	Timer/counter	8	NC	yes

(*a) NA refers to information not maintained by the microcomputer itself.

(*b) NC means not changed by RESET.

(*c) Figure 7H-1 shows the contents of the program status word.

(*d) You cannot use the S command to alter STACK or RETURN directly;
however, these values depend on bits 0--2 of the PSW, which can be
directly altered using S.

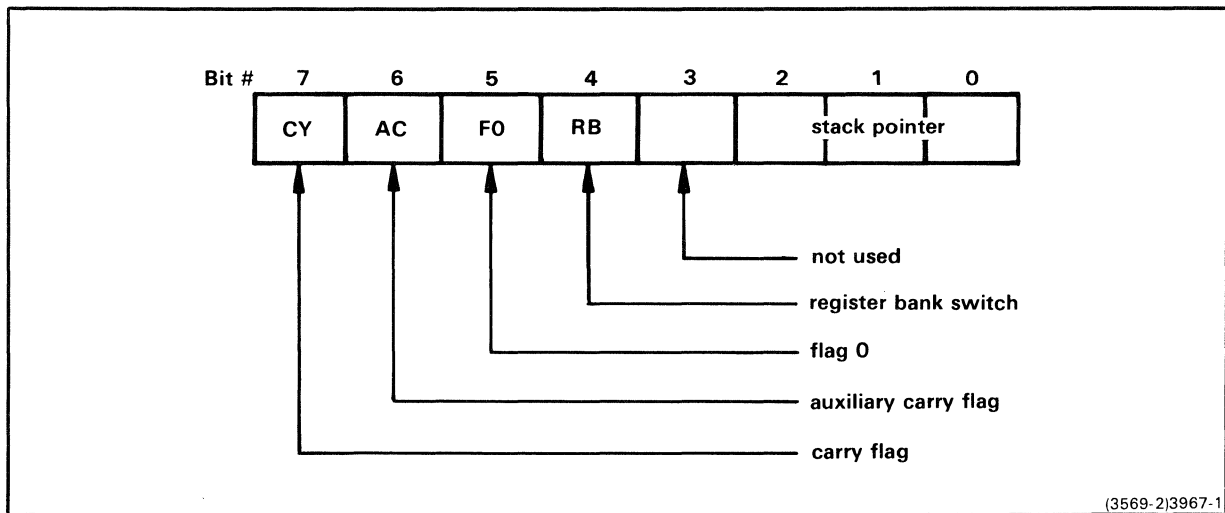


Fig. 7H-1. 8048/8021/8041A/8022 program status word.

DS --- Sample Status Display

The DS (Display Status) command displays the status and register contents of the emulator. All numbers in the DS display are hexadecimal. Here is an example of a DS display line produced by the 8048/8021/8041A/8022 emulator when it is used to emulate an 8048:

> DS <CR>

PC	CHIP	EPM	A	PSW	TR	RB		REGISTERS
010A	8048	Y	11	00	00	0		RO-R7= 19 36 07 A6 FD 00 00 00
								AO-A7= 86 00 2F 40 49 09 7D 00
IIP	EI	TI	TC	MB	STACK	RETURN	TF	FO
N	0	0	0	0	08	1F06	0	0
							F1	AN
								STF
								DMA
								IBF
								OBF
								FLG

The DS display differs slightly for the 8021, 8022, and 8041A. Table 7H-3 explains the symbols displayed by DS.

For the 8048/8021/8041A/8022 emulator, the short and long forms of the DS display are the same: DS -L gives the same display as DS.

The RESET command sends a hardware reset signal to the emulating microcomputer. The "Value After RESET" column of Table 7H-3 indicates which registers are affected by the RESET command.

> DS <CR>

```

PC      CHIP EPM  A PSW TR RB          REGISTERS
090A 8048  Y  02  FF 40  1      R0-R7= 86 00 2F 40 49 09 7D 00
                                     A0-A7= 19 36 07 A6 FD 00 FF 00

IIP EI TI TC MB STACK RETURN TF F0 F1 AN STF DMA IBF OBF FLG
  N  1  1  2  1   16   8208   1  1  1

```

> RESET <CR>

> DS <CR>

```

      |                                     | register bank 0 becomes the
      |                                     | current register bank
      v                                     v
PC    CHIP EPM A PSW TR RB               v          REGISTERS
0000  8048 Y  02  08 40  0             R0-R7= 19 36 07 A6 FD 00 FF 00
                                           AO-A7= 86 00 2F 40 49 09 7D 00

IIP EI TI TC MB STACK RETURN TF FO F1 AN STF DMA IBF OBF FLG
 N   ^  ^  ^  ^  ^     ^      ^      ^  ^  ^
 |   |  |  |  |       |      |      |  |  |

```

The arrows show the changed registers.

DI --- Sample Disassembled Code

The DI (DISassemble) command translates object code in memory into assembly language instructions. DI displays object code, assembly language mnemonics, and operands. Here is an example of 8048 DI output:

> DI 100 10E <CR>

LOC	INST	MNEM	OPER
000100	B932	MOV	R1,#32
000102	BF05	MOV	R7,#05
000104	27	CLR	A
000105	61	ADD	A,@R1
000106	19	INC	R1
000107	EF05	DJNZ	R7,#05 0105
000109	AA	MOV	R2,A
00010A	23F7	MOV	A,#F7
00010C	00	NOP	
00010D	A3	MOVP	A,@A
00010E	00	NOP	

=====

				--- operand(s): address, registers, or data
				being operated on
				----- instruction mnemonic
				----- machine language instruction
				----- address of instruction

TRA --- Sample TRAcE Display

The TRAcE command establishes the conditions for displaying trace lines during program execution. For the 8021 and 8022, the TRA -L display is the same as the default (TRA -S) display. For the 8048/8049 and the 8041A, the TRA -L display includes the contents of the alternate registers (A0--A7) in addition to the TRA -S information.

NOTE

When TRAcE conditions have been set, the emulator runs at slower than normal processing speeds and RTPA breakpoints are suppressed.

Here is an example of 8048 TRAcE output:

SERVICE CALLS

Service calls (SVCs) enable your program to use many system capabilities of your 8540, 8550, or 8560 while your program is running on the emulator processor.

An SVC is invoked with the following 8048/8021/8041A/8022 instruction sequence:

```

NOP
MOVP A,@A
NOP

```

The value in the accumulator at the time of the SVC directs the system to a specified memory address called the SRB pointer (which points to the SRB -- the Service Request Block). The SRB pointer tells the system where to find the data (stored in the SRB) that informs the system which function to perform. Refer to the Service Calls section of your System Users Manual for an explanation of service calls, service request blocks, and SRB pointers.

Your program can point to eight SRBs at any one time. (Under certain circumstances, your program can store new addresses in the SRB pointers as it executes. See the following discussion, "SVC Design Features.") Table 7H-4 shows the default addresses for the eight SRB pointers. These addresses and their associated accumulator values can be altered with the SVC command to suit your program requirements. See the Command Dictionary section of your System Users Manual for syntax and use of the SVC command.

SVCs in Modes 1 and 2

The 8048/8021/8041A/8022 emulator does not support SVCs in emulation modes 1 or 2.

Table 7H-4
8048/8021/8041A/8022 Service Calls

SVC Number	Value in Accumulator	Default SRB Pointer Location
1	F7	40,41
2	F6	42,43
3	F5	44,45
4	F4	46,47
5	F3	48,49
6	F2	4A,4B
7	F1	4C,4D
8	F0	4E,4F

SVC Demonstration

Figure 7H-2 lists an 8048/8021/8041A/8022 program that uses four SVC functions: Assign Channel, Read ASCII, Write ASCII, and Abort. The program's algorithm is explained in the Service Calls section of your System Users Manual, which demonstrates a version of the program written in 8085A assembly language. You can perform a parallel demonstration with the 8048/8021/8041A/8022 emulator and A Series Assembler using the program in Fig. 7H-2.

NOTE

The program shown in Fig. 7H-2 is written for an A Series assembler. To make this acceptable for a B Series assembler (as required by an 8560), change each double quote (") to a single quote (').

```

; SSSSS V V CCCCC
; S V V C
; SSSSS V V C DEMONSTRATION. 8048 EMULATOR
; S V V C
; SSSSS V CCCCC
; ORG 40H ; BEGINNING OF SRB VECTOR
; BYTE HI(SRB1FN),LO(SRB1FN)
; BYTE HI(SRB2FN),LO(SRB2FN)
; BYTE HI(SRB3FN),LO(SRB3FN)
; BYTE HI(SRB4FN),LO(SRB4FN)
; BYTE HI(SRB5FN),LO(SRB5FN)
; END OF SRB VECTOR
; ORG 50H ; SET UP SRB AREAS
; SRB1 = ASSIGN "CONI" TO CHANNEL 0
SRB1FN BYTE 10H ; ASSIGN
; BYTE 00H ; TO CHANNEL 0
SRB1ST BLOCK 01H ; STATUS RETURNED HERE
; BLOCK 02H ; BYTES 4 AND 5 NOT USED
; BYTE 05H ; LENGTH OF "CONI"+<CR>
; BYTE HI(CONI) ; POINTER TO
; BYTE LO(CONI) ; "CONI"+<CR>
; END OF SRB1
; SRB2 = ASSIGN "LPT" TO CHANNEL 1
SRB2FN BYTE 10H ; ASSIGN
; BYTE 01H ; TO CHANNEL 1
SRB2ST BLOCK 01H ; STATUS RETURNED HERE
; BLOCK 02H ; BYTES 4 AND 5 NOT USED
; BYTE 04H ; LENGTH OF "LPT"+<CR>
; BYTE HI(LPT) ; POINTER TO
; BYTE LO(LPT) ; "LPT"+<CR>
; END OF SRB2
; SRB3 = READ ASCII LINE FROM CONI (CHANNEL 0)
SRB3FN BYTE 01H ; READ ASCII
; BYTE 00H ; FROM CHANNEL 0
SRB3ST BLOCK 01H ; STATUS RETURNED HERE
; BLOCK 01H ; BYTE 4 NOT USED
; BLOCK 01H ; BYTE COUNT RETURNED HERE
; BYTE 00H ; 256 BYTES IN OUR BUFFER
; BYTE HI(BUFFER) ; POINTER TO
; BYTE LO(BUFFER) ; OUR BUFFER
; END OF SRB3
; SRB4 = WRITE ASCII LINE TO LPT (CHANNEL 1)
SRB4FN BYTE 02H ; WRITE ASCII
; BYTE 01H ; TO CHANNEL 1
SRB4ST BLOCK 01H ; STATUS RETURNED HERE
; BLOCK 01H ; BYTE 4 NOT USED
; BLOCK 01H ; BYTE COUNT RETURNED HERE
; BYTE 00H ; 256 BYTES IN OUR BUFFER
; BYTE HI(BUFFER) ; POINTER TO
; BYTE LO(BUFFER) ; OUR BUFFER
; END OF SRB4
; SRB5 = ABORT (CLOSE ALL CHANNELS AND TERMINATE)
SRB5FN BYTE 1FH ; ABORT
; BLOCK 07H ; BYTES 2 THROUGH 8 NOT USED
; END OF SRB5

```

3967-10

Fig. 7H-2. 8048 SVC demonstration program listing (part 1 of 2).


```

BUFFER  BLOCK  100H          ; OUR I/O AREA
CONI     ASCII  "CONI"        ; ASCII OF "CONI"
        BYTE   0DH           ; + <CR>
LPT      ASCII  "LPT"         ; ASCII OF "LPT"
        BYTE   0DH           ; + <CR>
;        END OF DATA DEFINITIONS
;
; BEGINNING OF EXECUTABLE CODE
START    ORG     10H          ; ENTRY POINT INTO PROGRAM
        MOV     A,#0F7H       ; CALL SVC1
        NOP                     ; TO ASSIGN "CONI"
        MOVP    A,@A          ; TO CHANNEL 0
        NOP                     ;
        MOV     A,#LO(SRB1ST) ; CHECK THE STATUS
        MOVP    A,@A          ; TO SEE IF ALL WENT WELL
        JNZ     ABORT         ; NO? STOP EVERYTHING
        MOV     A,#0F6H       ; CALL SVC2
        NOP                     ; TO ASSIGN "LPT"
        MOVP    A,@A          ; TO CHANNEL 1
        NOP                     ;
        MOV     A,#LO(SRB2ST) ; CHECK THE STATUS
        MOVP    A,@A          ; TO SEE IF ALL WENT WELL
        JNZ     ABORT         ; NO? STOP EVERYTHING
LOOP     MOV     A,#0F5H       ; CALL SVC3
        NOP                     ; TO READ A LINE
        MOVP    A,@A          ; FROM "CONI"
        NOP                     ; INTO THE BUFFER
        MOV     A,#LO(SRB3ST) ; CHECK THE STATUS
        MOVP    A,@A          ; TO SEE IF ALL WENT WELL
        JNZ     ABORT         ; NO? STOP EVERYTHING
        MOV     A,#0F4H       ; CALL SVC4
        NOP                     ; TO WRITE THE LINE
        MOVP    A,@A          ; FROM THE BUFFER
        NOP                     ; TO "LPT"
        MOV     A,#LO(SRB4ST) ; CHECK THE STATUS
        MOVP    A,@A          ; TO SEE IF ALL WENT WELL
        JZ      LOOP          ; YES? BACK TO READ ANOTHER LINE
        ; NO? FALL THROUGH TO TERMINATION
ABORT    MOV     A,#0F3H       ; CALL SVC5
        NOP                     ; TO DO THE ABORT
        MOVP    A,@A          ;
        NOP                     ;
        END     START         ; END OF PROGRAM

```

3967-11

Fig. 7H-2. 8048 SVC demonstration program listing (part 2 of 2).

This program shows the use of four 8550 service calls. The program's algorithm is explained in the Service Calls section of your System Users Manual. The program accepts a line of ASCII characters from the system terminal; then, when it receives a RETURN character, the program writes the line to the line printer and accepts another line. (On the 8550, output to the line printer is buffered. No text is printed until the line printer buffer in the 8501 becomes full or the program ends.) To terminate the program, enter a CTRL-Z while the program is waiting for input.

SVC DESIGN FEATURES

SVC data blocks (Service Request Blocks, SRB pointers, and I/O buffers) can reside in either of the following address ranges:

0000--0FFF program memory (internal and external)
2000--20FF external data memory

SVC data blocks cannot reside in internal data memory. If you are emulating an 8021, 8022, or 8041A, external data memory is not available, and program memory is limited to 400H bytes (for the 8021 and 8041A) or 800H bytes (for the 8022). For more information, refer to Figure 7H-3 and the discussion of memory mapping under the heading "Special Considerations".

If you are emulating an 8048/8049, programming considerations may require you to put some SVC data blocks in external data memory. For example, suppose you have a program that places information in an I/O buffer, and then uses an SVC to write out the contents of the buffer. If the information is read directly into the buffer and undergoes no further change (as in the preceding SVC demonstration program), the buffer can reside in program memory. Otherwise the buffer must reside in external data memory, because there are no 8048/8021/8041A/8022 instructions that write to program memory.

For information on allocating code to different types of memory, refer to the heading "Designating Memory Areas in Assembly Language Programs" in the Special Considerations subsection.

Moving Data Into and Out of Program Memory

The 8021, 8041A, and 8022 cannot access external memory. To provide a one-byte output buffer for these microcomputers, DOS/50 and OS/40 allow your program to change the contents of byte 0 of program memory. The following instruction sequence is used to move the contents of the accumulator to program memory location 0:

```
NOP
MOVX @R1,A
```

The following instruction sequence is used to move the contents of program memory location 0 to the accumulator:

```
NOP
MOVX A,@R1
```

NOTE

Whenever a NOP instruction is followed by a MOVX instruction, DOS/50 or OS/40 process the instruction sequence as an access to program memory location 0. The contents of register R1 and the selected register bank are ignored.

The NOP--MOVX sequences may be used when emulating any of the microcomputers in the 8048 family. However, before you execute these instruction sequences, make sure your 8550 or 8540 is in emulation mode 0 and TRAcE is OFF. Avoid using these sequences unless you're accessing program memory location 0.

Example. The following assembler directives set up an output buffer in bytes 0 and 1 of program memory. Your program can write to byte 0 using the NOP--MOVX instruction sequence. Byte 1 contains a RETURN character to mark the end of the buffer.

```
                ORG      0      ; BYTE 0 OF PROGRAM MEMORY
CHARBUF BLOCK 1      ; USED AS AN OUTPUT BUFFER.
                BYTE     0DH    ; BUFFER ENDS IN <CR>.
```

SPECIAL CONSIDERATIONS

The emulating microcomputer for the 8048/8021/8041A/8022 emulator is contained at all times on the prototype control probe, rather than on the emulator processor module (as with most other emulators). The 8048/8021 prototype control probe contains an 8039 microcomputer, and the 8041A and 8022 prototype control probes each contain an 8035 microcomputer. Since the emulating microcomputer is not the same as the microcomputer being emulated, certain special considerations must be noted here.

Other considerations of interest to users are also outlined in this subsection. First, considerations common to all microcomputers in the 8048 family are described. Then, considerations that apply only to certain devices are described.

CLOCK RATE

The frequency of the emulator clock, used in emulation mode 0, is 5.0 MHz. Table 7H-5 gives the maximum allowable frequencies for the prototype's clock, used in modes 1 and 2.

NOTE

For all microcomputers, the prototype clock frequency must be at least 2.0 MHz in order to fulfill the clock requirement of the emulating 8035 or 8039 microcomputer.

Table 7H-5
Maximum Allowable Clock Frequencies

	Maximum Allowable Clock Frequency			
	8049, 8039 (*a)	8048, 8648, 8748, 8035, 8039-6	8041A, 8741A	8022
Emulation Mode				
Mode 1 (Mapped to 8550)	11.0 MHz (*b)	6.0 MHz	6.0 MHz	3.58 MHz
Mode 1 (Mapped to Prototype) and Mode 2	11.0 MHz	6.0 MHz	External (prototype) memory is not addressable by these microcomputers.	

(*a) To use clock frequencies above 6.0 MHz, the configuration switch in the prototype control probe's interface assembly must be set as prescribed in the 8048/8021/8041A/8022 Emulator Processor and Prototype Control Probe Installation Service Manual.

(*b) In emulation mode 1 with memory mapped to 8550 program memory, the clock frequency is divided by 2 to produce an effective clock rate of 5.5 MHz.

MEMORY MAPPING

Figure 7H-3 illustrates the memory layouts for the microcomputers in the 8048 family. You can access all three 8048/8021/8041A/8022 memory areas (program memory, internal data memory, and external data memory) using standard DOS/50 or OS/40 memory manipulation commands such as D, EX, F, LO, P, and SEA.

NOTE

In this Emulator Specifics section, the term program memory refers to the on-board ROM and/or external memory in which 8048/8021/8041A/8022 program instructions are stored. The term 8550/8540 program memory refers to the RAM in the 8550 or 8540 that may be used as a substitute for memory in the microcomputer and/or prototype. Throughout the other sections of your System Users Manual, the term program memory refers to 8550 or 8540 program memory.

Program Memory. The 8048 contains 1K of on-board ROM, and the 8049 contains 2K of on-board ROM. The 8048 and 8049 can each address 4K of external program memory.

The 8021 and 8041A each contain 1K of on-board ROM. The 8022 contains 2K of on-board ROM. The 8021, 8041A, and 8022 cannot access external memory.

The emulating microcomputer contains no on-board ROM. Program memory is emulated by bytes 0000--0FFF of 8550/8540 program memory. For example, the Dump command D 0 3F displays bytes 0--3F of program memory.

For the 8048/8049 only, program memory can be mapped to the prototype. To run a program that resides in external program memory on the prototype, three conditions must be satisfied:

- The prototype EA line (8048 input) must be asserted.
- In mode 1, the appropriate address ranges of program memory must be mapped to the prototype (MAP U).
- The switch pack on the 8048/8021 prototype control probe assembly must be set to enable external program memory. For information on how to set the switch, refer to your 8048/8021/8041A/8022 Emulator Processor Installation Manual.

NOTE

DOS/50 and OS/40 cannot write to program memory on the prototype. For example, you cannot use the LO or MOV command to load instructions into program memory on the prototype.

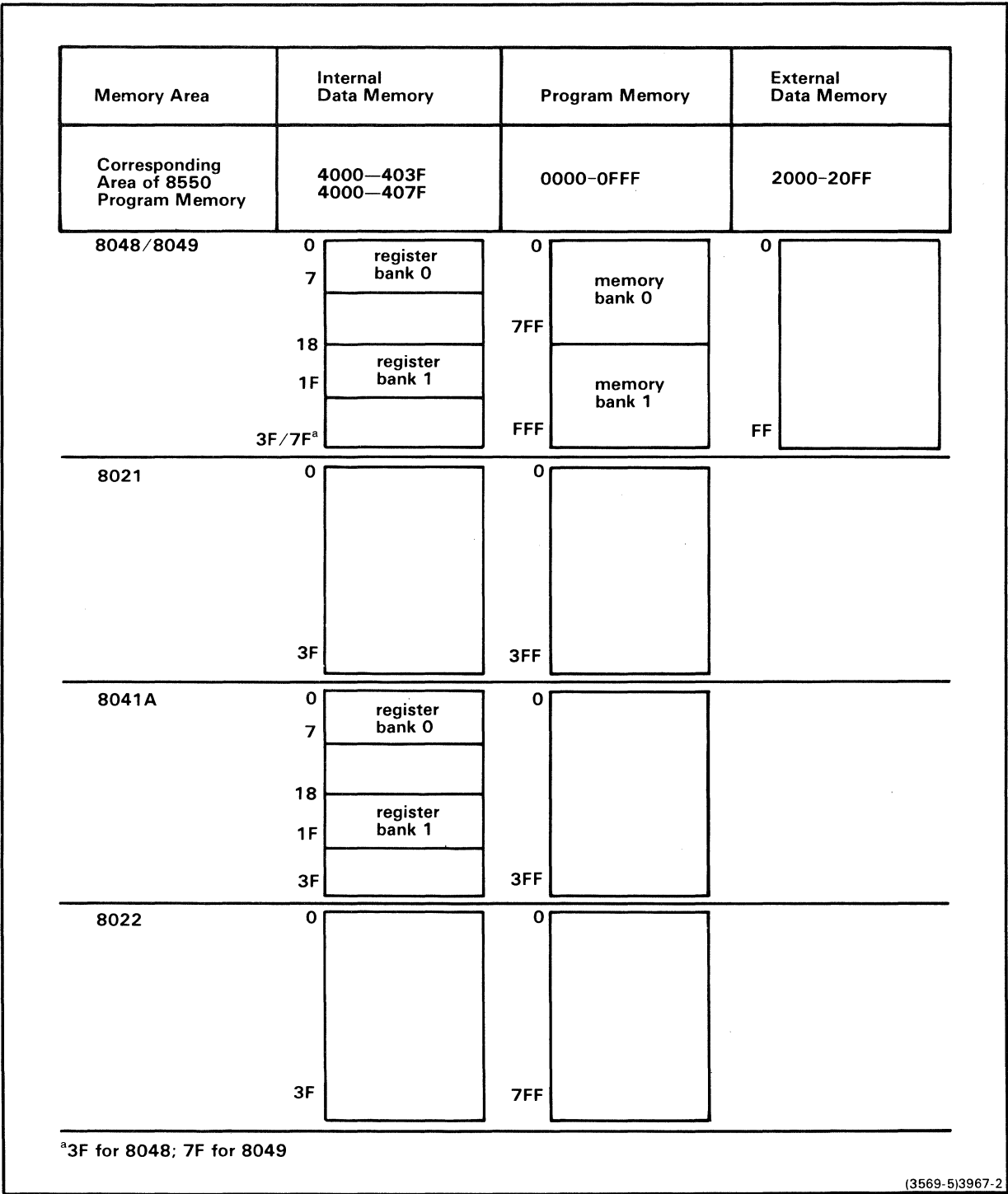


Fig. 7H-3. 8048/8021/8041A/8022 memory layout.

Internal Data Memory. The 8049 contains 128 bytes of internal RAM ("internal data memory"). Each of the other microcomputers (the 8048, 8021, 8041A, and 8022) contains 64 bytes of internal RAM.

Although this RAM actually resides in the emulating microcomputer, it can be treated as if it occupied bytes 4000--403F of 8550/8540 program memory (for the 8049, bytes 4000--407F.) For example, the command D 4000 4007 displays the contents of registers R0--R7, which occupy bytes 0--7 of internal data memory. You can access the registers in internal data memory using the S (Set) command as well as the standard DOS/50 or OS/40 memory manipulation commands.

NOTE

To access 8550/8540 program memory in the range 4000--407F, first transfer the contents of locations 4000-407F to another area of 8550/8540 program memory:

> MOV PP 4000 407F address <CR> (MOVE Program to Program),

then use standard operating system commands.

To transfer the contents of internal data memory to 8550/8540 program memory, enter:

> MOV UP 4000 407F address <CR> (MOVE User to Program).

NOTE

When any 8048-family microcomputer other than the 8049 is being emulated, bytes 4040--407F of internal data memory contain 0FFH, and can only be accessed by the D command.

Internal data memory cannot be mapped to the prototype.

External Data Memory. Using the MOVX instruction, the 8048/8049 can address 256 bytes of external RAM. This external data memory is emulated by bytes 2000--20FF of 8550 or 8540 program memory. For example, the command D 2000 20FF displays the contents of external data memory. The command MAP U 2000 20FF maps all of external data memory to the prototype.

The 8021, 8041A, and 8022 cannot access external memory.

NOTE

To correctly emulate a microcomputer, restrict programming to within its memory limitations, as shown in Fig. 7H-4. Errors in loading or execution will result if your program attempts to access locations outside the appropriate memory areas.

Designating Memory Areas in Assembly Language Programs

The DOS/50 or OS/40 LO command can be used to load data into all three 8048/8021/8041A/8022 memory areas. To load a data item into internal data memory, locate the data item in the range 4000--403F (for the 8049, 4000--407F). To load a data item into external data memory, locate the data item in the range 2000--20FF.

Modules That Do Not Require Linking. To locate the data in the proper range, use ORG directives. In the following example, after the module is assembled and loaded, bytes 18--1F of internal data memory are set to zero and bytes 0--4 of external data memory contain the word "ERROR".

```

ORG      4018H          ; INTERNAL DATA MEMORY LOCATION 18
BYTE     0,0,0,0,0,0,0,0 ; 8 ZEROS
ORG      2000H          ; EXTERNAL DATA MEMORY LOCATION 0
ASCII    "ERROR"        ; 5 LETTERS
END

```

Modules That Must Be Linked Before Loading. Some modules must be linked before they are loaded. For example, program modules that use symbolic debug are linked in order to access symbols from the object file. In the following example, a new section is defined for each area of memory. The ORG directives are relative to the beginning of each section. Then, when you invoke either the A Series or B Series linker, include the LOCATE command option for each section. (The demonstration run program, later in this section, includes examples of linker usage.)

```

SECTION INTERNAL          ; DEFINE THE INTERNAL MEMORY SECTION
ORG      18H              ; INTERNAL DATA MEMORY LOCATION 18
FIRST BYTE 0,0,0,0,0,0,0,0 ; 8 ZEROS
SECTION EXTERNAL          ; DEFINE THE EXTERNAL MEMORY SECTION
ORG      0H               ; EXTERNAL DATA MEMORY LOCATION 0
NEXT ASCII "ERROR"        ; 5 LETTERS
LIST     DBG
END      FIRST

```

For more information about the linker, see your A Series or B Series Assembler Users Manual.

EMULATION MODES8021/8041A/8022 Emulation Modes

Because the 8021, 8041A, and 8022 have no external memory to map to the prototype, only emulation modes 0 and 1 are available for these microcomputers. In mode 0, the emulator clock is used and program I/O is handled through service calls (SVCs). In mode 1, SVCs are disabled and the prototype's clock and I/O facilities are used. In both modes, 8550/8540 program memory serves as a substitute for the microcomputer's program memory.

8048/8049 Emulation Modes

All three emulation modes are available for the 8048/8049: both program memory and external data memory can be mapped to the prototype in modes 1 and 2. If a break occurs while the microcomputer is in an interrupt service routine, the IIP (interrupt in process) flag is not reset, and the microcomputer does not return from the interrupt. When the microcomputer is restarted after the break, the only way to return from the interrupt is to complete the interrupt service routine (or enter the RESET command). Otherwise, the microcomputer will not accept additional interrupts, and the IIP flag will remain set.

When an interrupt and an instruction fetch on a breakpoint occur simultaneously, a false break line appears. The microcomputer performs the interrupt service routine, then returns and breaks.

NOP--MOVX INSTRUCTION SEQUENCE

Avoid using the NOP--MOVX command sequence except to access byte 0 of program memory. This design feature is described more fully under the heading "SVC Design Features" earlier in this section.

Normally, the MOVX instruction is illegal for the 8021, 8041A, and 8022 microcomputers. However, DOS/50 and OS/40 recognizes this MOVX sequence as valid for the specific purpose of accessing program memory location 0.

NOTE

Be sure that you have specified TRA OFF before attempting to use MOVX for this purpose.

TRA AND THE TIMER/COUNTER

Be sure that TRAcE is OFF before your program attempts to use the timer/counter. The timer/counter can interfere with the internal operations of DOS/50 or OS/40 when TRA selections are active.

8048/8049 SPECIAL CONSIDERATIONSPort 0 Latch

On an OUTL instruction, the 8048/8049 normally outputs data during S5 of the first cycle of the instruction. On ANL and ORL instructions, it outputs data during S4 of the second cycle. However, for all three instructions, the emulator outputs port 0 data during S1 of the second cycle of the instruction.

Port 2 Latch and External Program Memory Switch

A switch in the 8048/8021 prototype control probe enables the emulating microcomputer to access external program memory. The status of this switch affects activity on pins P20--P23, which output the four high-order bits of the program counter during instruction fetches to external program memory.

When this switch is on, pins P20--P23 behave as if external program memory is always being accessed, regardless of the value of the program counter and the state of the EA input. The address information on port 2 will always be output; the I/O information on port 2 will unlatch during every memory access, then relatch on the leading (rising) edge of the subsequent ALE. When an IN P2,A instruction is executed and the switch is on, the emulator will sample the P20--P23 pins regardless of whether they were previously programmed as "output." Any of the P20--P23 lines that are not driven by the prototype will appear as ones in the accumulator.

On an OUTL instruction, the 8048/8049 normally latches data during S5 of the first cycle of the instruction. On ANL and ORL instructions, it latches data during S4 of the second cycle. However, when the external program memory switch is off, the emulator latches P20--P23 I/O data during S3 of the second cycle of the next instruction.

8048/8021 Address Latch Enable (ALE)

The emulating microcomputer in the 8048/8021 prototype control probe is an 8039. The 8039 continuously issues the ALE signal to the prototype, even when it is executing 8550 or 8540 debug routines. The internal program addresses of these debug routines may be interspersed with your program's addresses.

Prototype Clock

The 8048/8021 prototype control probe treats the X1 clock pin as an input and the X2 pin as an output in order to drive a crystal circuit. However, Intel's revised specifications for the 8048, 8035, 8049, and 8039 microcomputers require that both pins be driven if the clock source is an external TTL clock. Therefore, if your prototype uses an external TTL clock, the prototype X2 driver should be disconnected while using the emulator.

8021 SPECIAL CONSIDERATIONS8048/8021 Address Latch Enable (ALE)

The emulating microcomputer in the 8048/8021 prototype control probe is an 8039. The 8039 continuously issues the ALE signal to the prototype, even when it is executing 8550 or 8540 debug routines. The internal program addresses of these debug routines may be interspersed with your program's addresses.

JMP and CALL Instructions

The 8021 is emulated by an 8039 microcomputer. The 8039 can address 2K of program memory, while the 8021 has only 1K. Thus, whenever you use a JMP or CALL instruction with the 8021, the op code must be in the range 04, 14, ..., 64, 74. For the 8021, the following op codes are illegal: 84, 94, A4, B4, C4, D4, E4, and F4. If the emulator attempts to execute one of these illegal codes, no error indication will appear in your TRAcE display; the program will simply halt. The only way to determine the cause of the halt is to inspect the op codes in your listing.

8041A SPECIAL CONSIDERATIONS

JMP and CALL Instructions

The 8041A is emulated by an 8035 microcomputer. The 8035 can address 2K of program memory, while the 8041A has only 1K. Thus, whenever you use a JMP or CALL instruction with the 8041A, the op code must be in the range 04, 14, ..., 64, 74. For the 8041A, the following op codes are illegal: 84, 94, A4, B4, C4, D4, E4, and F4. If the emulator attempts to execute one of these illegal codes, no error indication will appear in your TRAcE display; the program will simply halt. The only way to determine the cause of the halt is to inspect the op codes in your listing.

Input/Output Buffers

The 8041A has internal input/output buffers. On the 8041A prototype control probe, these buffers are located external to the emulating 8035 microcomputer. Thus, the buffer contents can be altered even when the emulator is not active. In such cases, the buffer flags would be affected, and the DS display would not show the correct emulator status.

RETR Instruction

In the 8041A prototype control probe, the F0 flag is placed on the F0 flag stack by an interrupt, but not by a subroutine call. Therefore, use a RET instruction to return from a subroutine. Use a RETR instruction only to return from an interrupt service routine.

8022 SPECIAL CONSIDERATIONS

RETI Instruction

The op code for the 8022 RETI instruction is 93H. However, the emulating 8035 microcomputer recognizes that op code as a RETR instruction. The 8035 RETR instruction performs three functions:

- allows interrupts (if interrupts are enabled);
- restores the program counter from the stack;
- restores the program status word (PSW) from the stack.

However, the 8022 RETI instruction only performs the first two functions. Thus, whenever your program includes a RETI instruction, the 8035 will automatically restore the PSW. However, your prototype 8022 will not restore the PSW.

Also note that the 8022 RETI instruction increments the program counter, while the 8035 RETR instruction does not. Thus, the possibility exists that your program could execute the same instruction twice upon return from an interrupt.

A-to-D Converter

The 8022 requires that the analog input be maintained at a constant voltage during the sample time. A SEL AN0 or SEL AN1 instruction changes the input voltage. Thus, the first RAD instruction after any SEL instruction will be inaccurate. Each succeeding RAD will be accurate.

Pullup Options

Two pullup options are available with the 8022. With the emulating 8035 microcomputer, these options are selected by a cuttable run (T1) and by switch settings (Port 0) on the 8022 prototype control probe. Refer to your 8048/8021/8041A/8022 Emulator Processor Installation Manual for information on how to set these pullup options.

EMULATOR TIMING

The emulating microcomputer resides in the prototype control probe, and the signals between the prototype and the emulating microcomputer are buffered. Therefore, some timing differences exist between the 8048/8021/8041A/8022 emulator and a microcomputer inserted directly into the prototype.

Table 7H-6 lists the probe/microcomputer timing differences for the 8048/8021 prototype control probe. Figure 7H-4 contains timing diagrams corresponding to the signals listed in that table.

Similarly, Tables 7H-7 and 7H-8 and Fig. 7H-5 give timing differences for the 8041A prototype control probe. Table 7H-9 and Fig. 7H-6 give timing differences for the 8022 prototype control probe.

Table 7H-6
Probe/Prototype Interface Delays for the 8048/8021

Signal		t(PHL) (ns)	t(PLH) (ns)
		Maximum	Maximum
ALE		20	20
PSEN		32	32
RD, WR		26	22
PROG		20	20
DB0--DB7 (*a)	t(1)---fetch cycle	90	90
(P00-P07)	t(2)---execute cycle	38	38
Prototype to CPU			
DB0--DB7	t(3)---address out	38	38
(P00-P07)	t(4)---external data	38	38
CPU to Prototype	out		
	t(5)---OUTL, ANL, ORL data out	(*b)	(*b)
P10--P17, P24--P27		2	2
P20--P23		(*c)	(*c)
T0 (*d) out/in		15	15
T1			182
INT		32	32
RST (t(PHL) for 8021, t(PLH) for 8048)		284	284
SS		32	32
CLK		79	79

(*a) $t(RD) = t(1,2) + t(\text{prototype memory access})$

(*b) OUTL, ANL, and ORL bus I/O information is latched during S1 of the second cycle of the instruction.

(*c) When external program memory is enabled (by a DIP switch in the prototype control probe), the maximum address delay is 32 ns. When external program memory is disabled, OUTL, ANL, and ORL information is latched during the S3 cycle following the next instruction fetch.

(*d) If the prototype clock frequency is greater than 6 MHz and memory is mapped to the 8550 or 8540, T0 out is divided by 2.

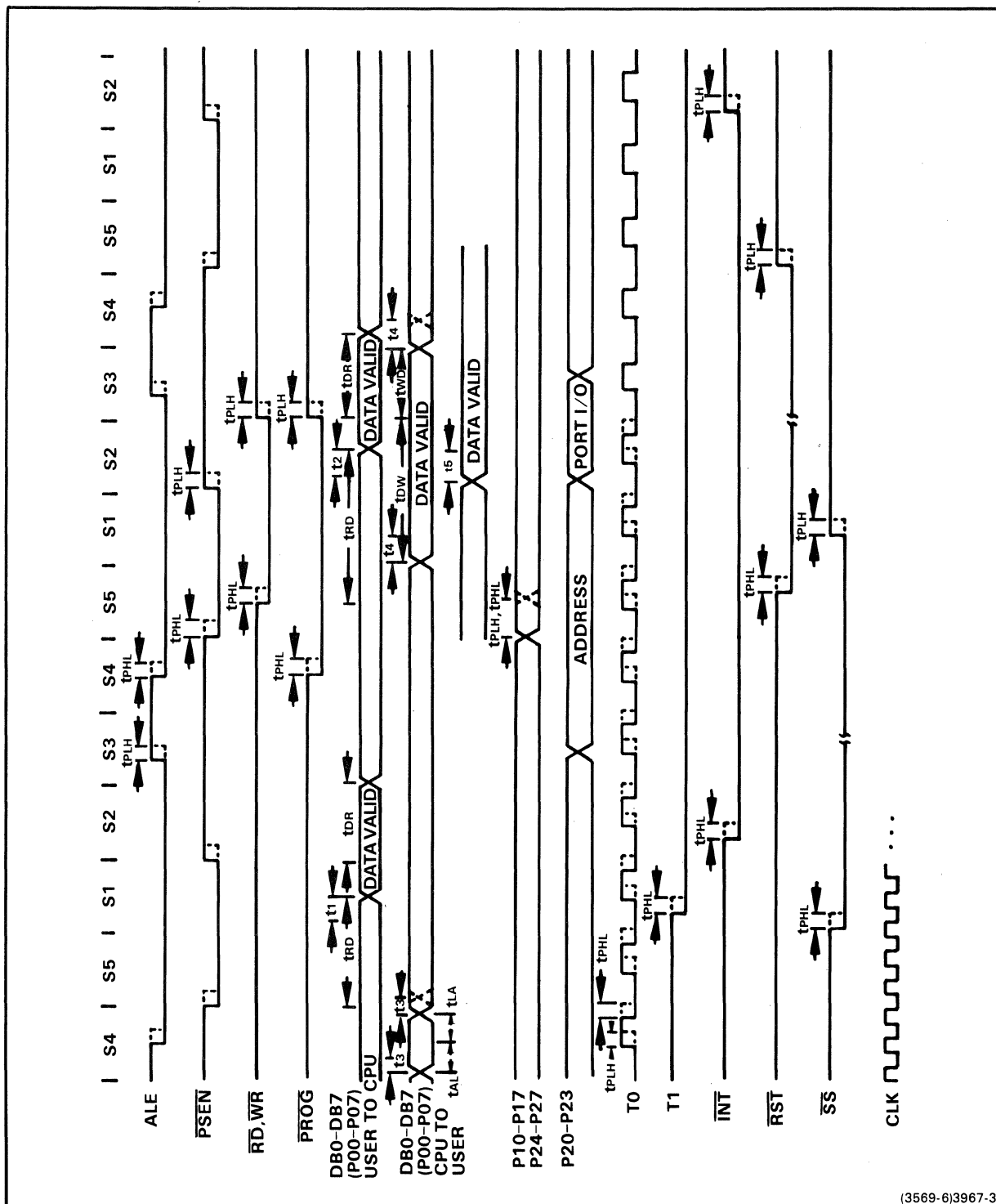


Fig. 7H-4. 8048/8021 timing diagrams.

These timing diagrams illustrate the signals described in Table 7H-6. For signals sent from the CPU to the prototype, solid lines represent timing at the CPU and dashed lines represent timing seen by the prototype. For signals sent from the prototype to the CPU, solid lines represent timing at the prototype and dashed lines represent timing seen by the CPU.

Table 7H-7
Probe/Prototype Interface Delays for the 8041A

Signal	t(PHL) (ns)	t(PLH) (ns)
	Maximum	Maximum
SYNC	20	20
PROG	20	20
T1		39
P10--P17	2	2
T0	45	34

Table 7H-8
Representative 8041A Probe/8041A Microcomputer Timing Differences

Symbol	Parameter	Microcomputer		Probe		Units
		Min.	Max.	Min.	Max.	
t(ACC)	DACK fall to WR or RD	0		54		ns
t(CAC)	RD or WR to DACK rise	0		71		ns
t(ACD)	DACK fall to data valid		225		225	ns
t(CRQ)	RD or WR to DRQ cleared		200		200	ns
t(AW)	CS, A0 setup to WR fall	0		0		ns
t(WA)	CS, A0 hold after WR fall	0		24		ns
t(WW)	WR pulse width	250		250		ns
t(DW)	Data setup to WR rise	150		150		ns
t(WD)	Data hold after WR rise	0		70		ns

Timing assumptions: CPU timing reference is Intel Peripheral Design Handbook, published by Intel Corp., 1979.

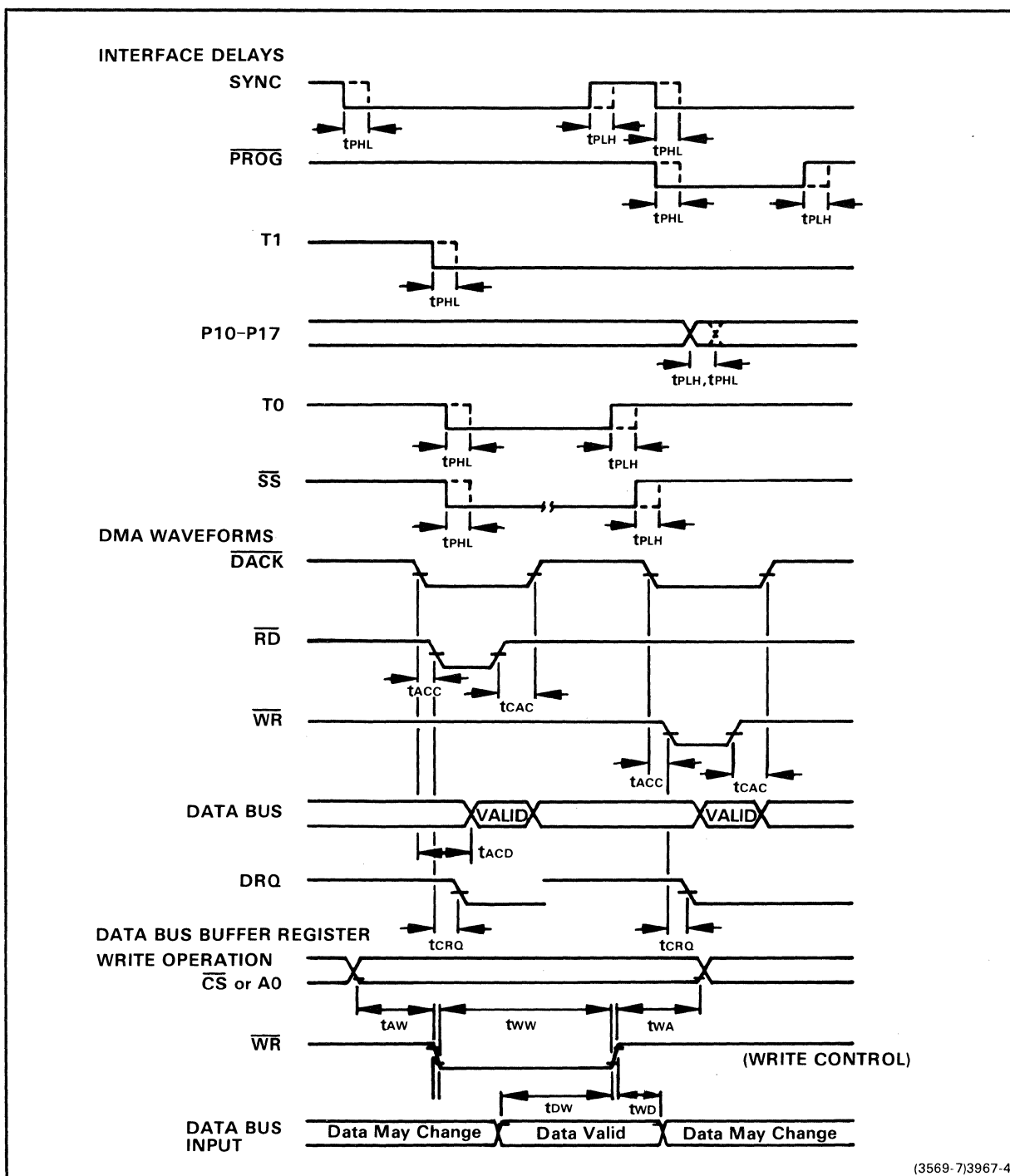


Fig. 7H-5. 8041A timing diagrams.

These timing diagrams illustrate the signals described in Tables 7H-7 and 7H-8. Dashed lines represent signal delays between the CPU and the prototype.

Table 7H-9
Probe/Prototype Interface Delays for the 8022

Signal		t(PLH) Maximum	t(PHL) Maximum	Units
=====				
ALE		34	46	ns
P00--P07	CPU to prototype	87	91	ns
P00--P07	prototype to CPU	1.3	1.3	us

P10--P17	t(1)---CPU to prototype	2	2	ns
	t(2)---prototype to CPU	2	2	ns

P20--P23	t(3)---A to P2	18	24	ns
(*a, *b)	t(4)---P2 to A	18	24	ns

PROG		18	24	ns
T0		24	24	ns
T1		182	182	ns
ANO, AN1 (*c)		444	444	ns
XTAL1		33	45	ns
RESET		229	229	ns

(*a) Inputs must be present until read by an input instruction (Intel Specification).				
(*b) For OUTL, ORL, and ANL instructions, data will be valid before ALE following the next instruction fetch.				
(*c) Input capacitance: 37 pf maximum.				

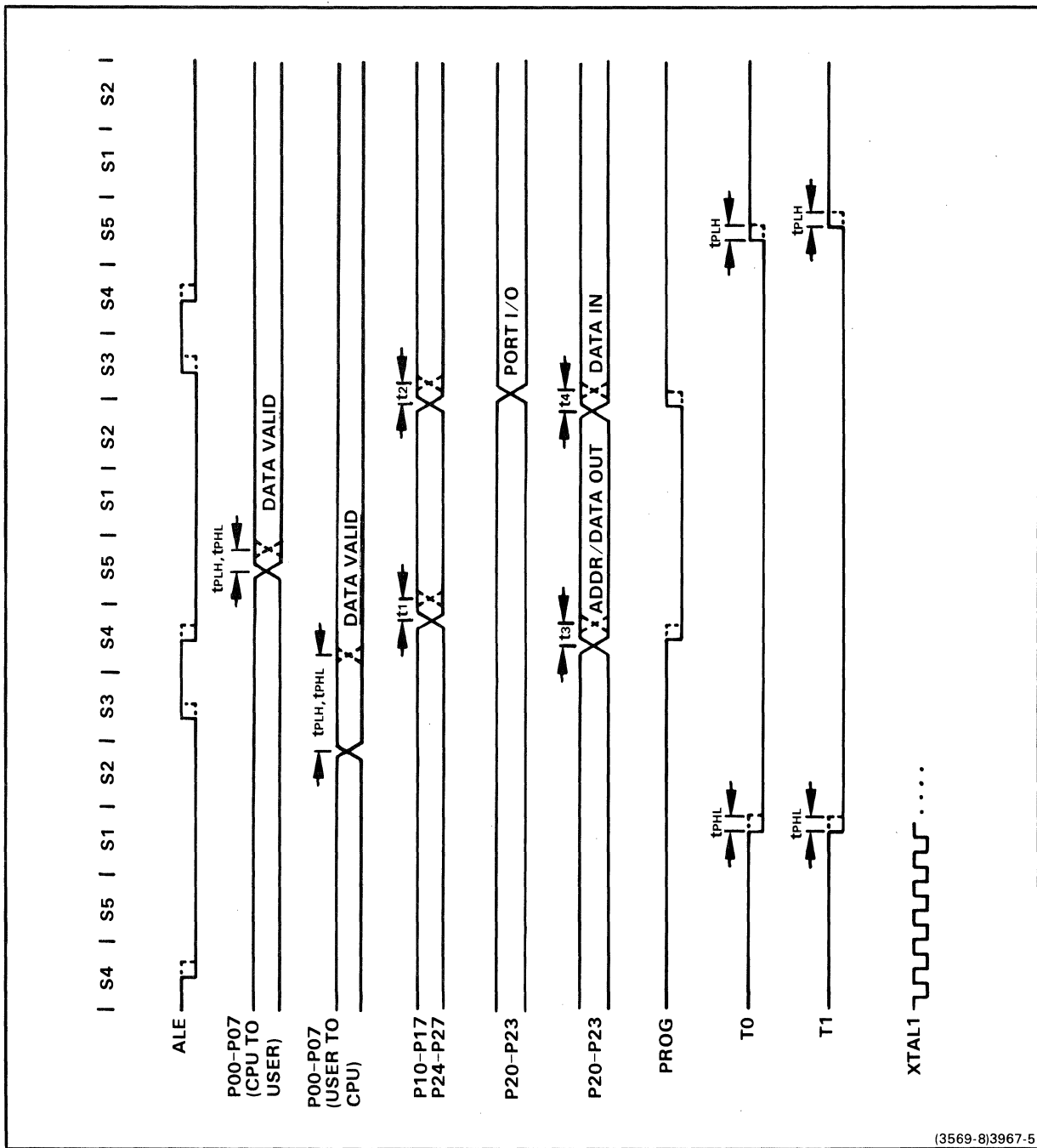
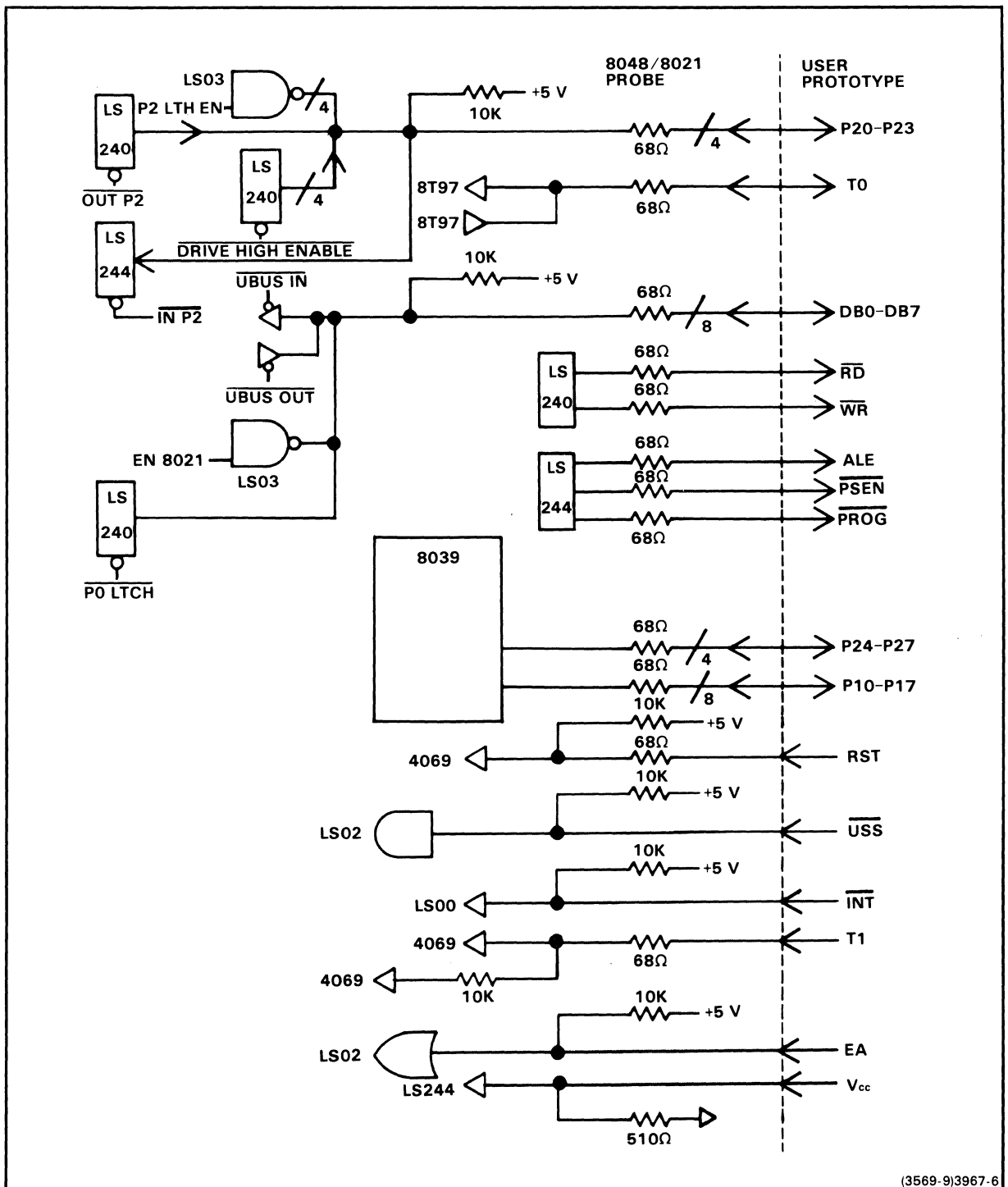


Fig. 7H-6. 8022 timing diagrams.

These timing diagrams illustrate the signals described in Table 7H-9. Dashed lines represent signal delays between the CPU and the prototype.

PROBE/PROTOTYPE INTERFACE DIAGRAMS

Figures 7H-7, 7H-8, and 7H-9 are block diagrams of the probe/prototype interfaces for the 8048/8021, 8041A, and 8022 prototype control probes, respectively.



(3569-9)3967-6

Fig. 7H-7. Block diagram of 8048/8021 probe/prototype interface.

This figure provides a functional overview of signal buffering between the prototype and the emulating microcomputer on the 8048/8021 prototype control probe. A more detailed circuit description can be found in the 8048/8021/8041A/8022 Emulator Service Manual.

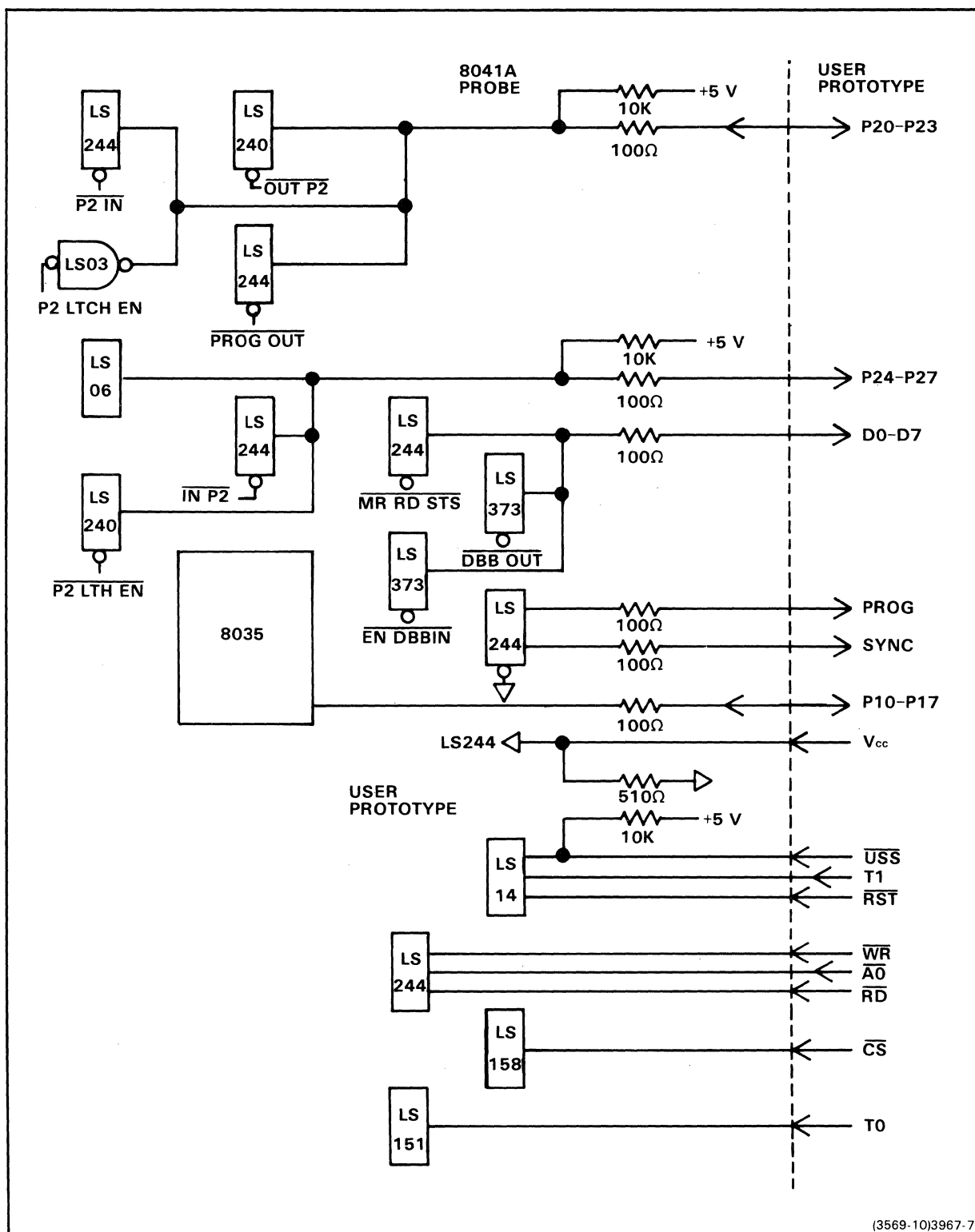


Fig. 7H-8. Block diagram of 8041A probe/prototype interface.

This figure provides a functional overview of signal buffering between the prototype and the emulating microcomputer on the 8041A prototype control probe. A more detailed circuit description can be found in the 8048/8021/8041A/8022 Emulator Service Manual.

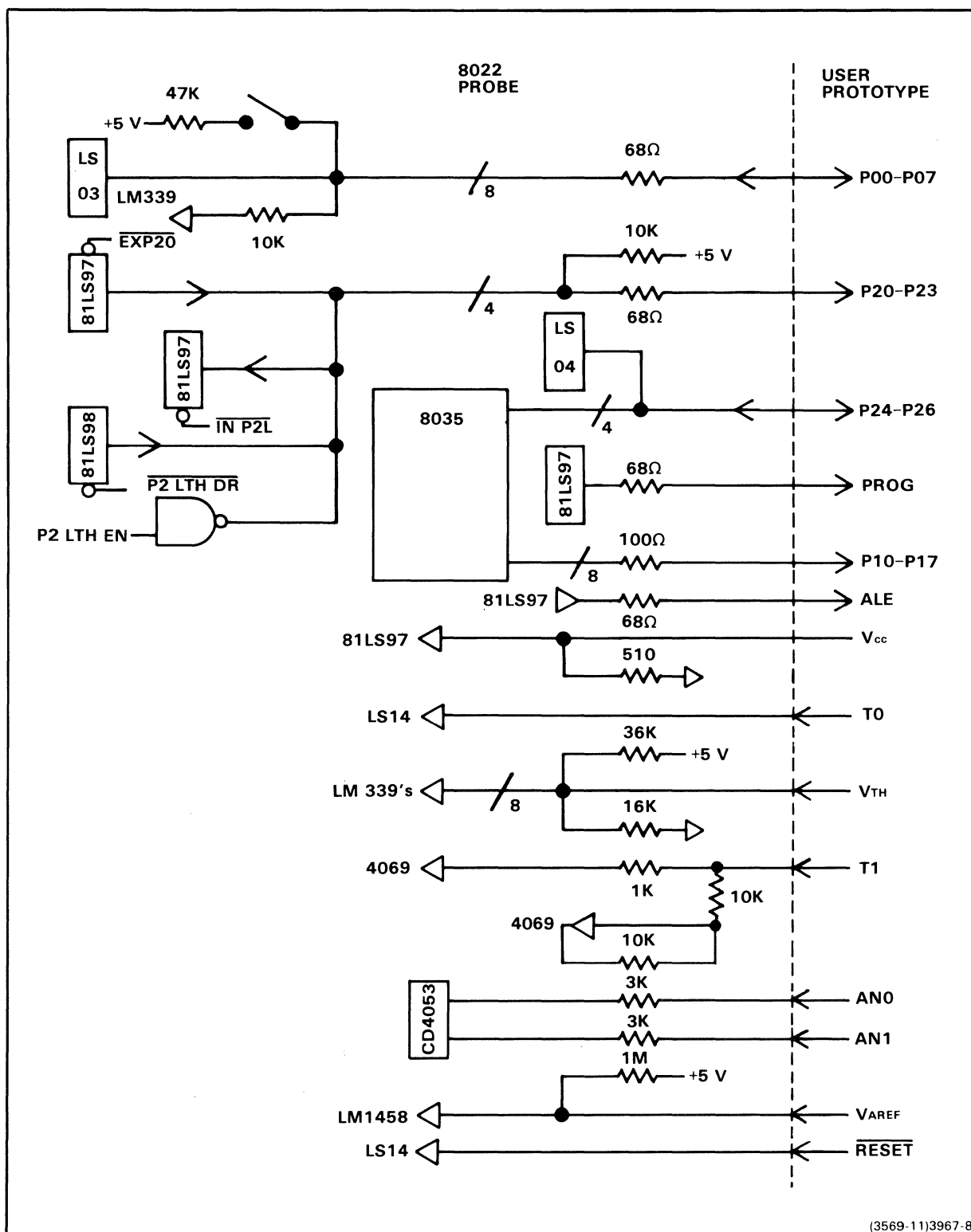


Fig. 7H-9. Block diagram of 8022 probe/prototype interface.

This figure provides a functional overview of signal buffering between the prototype and the emulating microcomputer on the 8022 prototype control probe. A more detailed circuit description can be found in the 8048/8021/8041A/8022 Emulator Service Manual.

INSTALLING YOUR 8048/8021/8041A/8022 EMULATOR SOFTWARE8540 FIRMWARE INSTALLATION PROCEDURE

The ROMs that contain the control software for your 8048/8021/8041A/8022 emulator must be installed in your 8540's System ROM Board. Refer to your Emulator Installation Manual for instructions on installing these ROMs.

8550 INSTALLATION PROCEDURE

This subsection describes how to install the control software for your 8048/8021/8041A/8022 emulator. To complete this installation procedure, you need the following items:

- an 8550 system (with or without an 8048/8021/8041A/8022 emulator)
- a DOS/50 system disk with a write-enable tab over the write-protect slot
- an 8048/8021/8041A/8022 emulator software installation disk with no write-enable tab.

This procedure takes about five minutes.

Start up and Set the Date

Turn on your 8550 system. (For start-up instructions, refer to the paragraph "Start Up the 8550 and Its Peripherals" in the Learning Guide section of your System Users Manual.) Place your system disk in disk drive 0 and shut the drive 0 door. When you see the > prompt on your system terminal, place your installation disk in disk drive 1 and shut the drive 1 door.

Use the DAT command to set the date and time. For example, if it is 11:05 am on April 1, 1983, type:

```
> DAT 01-APR-83/11:05 <CR>
```

The system will use this information when it sets the Creation Time attribute of each file copied from your installation disk.

Install the Software

The command file INSTALL2, which is used to install the software, resides on the installation disk.

NOTE

If your system disk contains DOS/50 Version 1, use the command file INSTALL instead of INSTALL2.

To execute this command file, type its filespec:

```
> /VOL/EMU.8048/INSTALL2 <CR>
```


DOS/50 responds with the following message:

```
* During this installation procedure, one or more of the
* following messages may appear.  IGNORE THESE MESSAGES:
*
*      Error 6E - Directory alteration invalid
*      Error 7E - Error in command execution
*      Error 1D - File not found
*
* If any OTHER error message appears, see your
* Users Manual for further instructions.
*
* If no other error message appears, you'll receive a
* message when the installation procedure is complete.
*
T,OFF
```

In the installation process, you may disregard error messages 6E, 7E, and 1D; these messages have no bearing on the success of the installation. However, if a message other than 6E, 7E, or 1D appears, take the following steps:

1. Make sure you are using the right disks.
2. Make sure your system disk has a write-enable tab.
3. Make sure there are at least 3 free files and 20 free blocks on your system disk.
4. Begin the installation procedure again.

If the installation procedure fails again, copy down the error message and contact your Tektronix service representative.

The "T,OFF" command suppresses subsequent output to your system terminal (except error messages) until INSTALL2 finishes executing. Within about five minutes, INSTALL2 will finish and your system terminal will display the following message:

```
*
* Your installation has been successfully completed.
>
```

Once your software is installed, you can:

- remove your disks and turn off your 8550 system, or
- install more software, or
- continue with the 8048/8021/8041A/8022 Emulator Demonstration Run that follows in this section. If you do this, you do not have to restart the system or reset the date and time.

NOTE

At this point, "NO.NAME" is the current user. To change the current user back to "yourname", enter USER,,yourname.

8048/8021/8041A/8022 DEMONSTRATION RUN

INTRODUCTION

This Demonstration Run shows you how to assemble, load, execute, and monitor a simple 8048 assembly language program on your 8540 or 8550. This program can be executed using any prototype control probe that is compatible with the 8048/8021/8041A/8022 emulator. In order to perform this demonstration, your 8048/8021/8041A/8022 emulator, prototype control probe, and control software must be installed in your 8540 or 8550.

Figure 7H-10 shows the source and object code for the demonstration program. If you have an 8550, the source code and object code for the demonstration program are provided on the installation disk that contains your 8048/8021/8041A/8022 emulator control software. This demonstration shows you how to assemble the program on your 8550. (If your system disk does not contain a 8048/8021/8041A/8022 assembler, you will have to skip that part of the demonstration.)

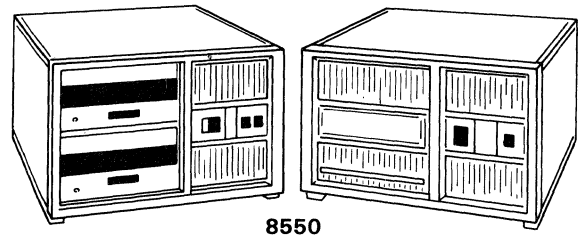
If you have an 8540/8560 system and your 8560 has a 8048/8021/8041A/8022 assembler installed, you can create and assemble the program on the 8560 and then download it to the 8540. This demonstration shows how.

If you have an 8540 that is connected to a host computer other than an 8560, we can't give you a specific list of commands for creating and assembling the program on your host (since we don't know what host you're using). However, Fig. 7H-11 gives the object code for the program in Extended Tekhex format. You can create the Tekhex file using your host's assembler or text editor, and then download the file to the 8540 via the 8540's optional COM interface.

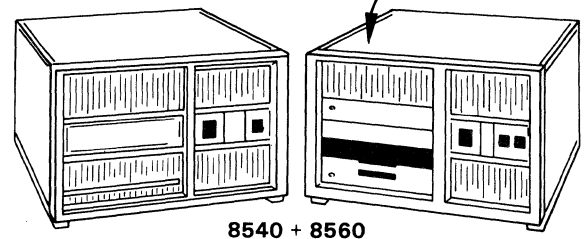
If none of these cases applies to you, you can patch the program into memory using the P command. This demonstration shows how.

Once the program is loaded or patched into memory, you can execute the program on your emulator.

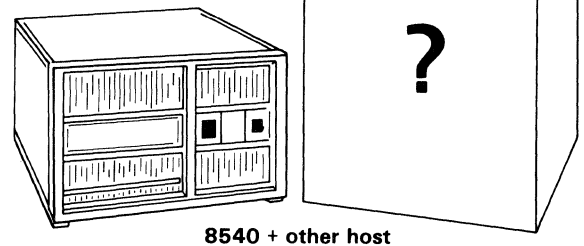
Case 1:



Case 2:



Case 3:



Case 4:

any other configuration

(3964-5)3967-9

NOTE

The 8540 commands shown in this demonstration can also be used for an 8550 that is connected to an 8560 or other host computer.

```

01 ;8048 DEMONSTRATION RUN PROGRAM
02 SECTION DEMO
03 000100 ORG 100H ;START PROGRAM CODE AT ADDRESS 100
04 000100 B932 START MOV R1,#TABLE ;SET TABLE POINTER
05 000102 BF05 MOV R7,#TSIZE ;SET PASS COUNTER
06 000104 27 CLR A ;CLEAR ACCUMULATOR
07 000105 61 LOOP ADD A,@R1 ;ADD BYTE FROM TABLE
08 000106 19 INC R1 ;POINT TO NEXT BYTE
09 000107 EF05 DJNZ R7,LOOP ;DECREMENT PASS COUNTER AND
10 ; LOOP IF NOT FIVE PASSES YET
11 000109 AA MOV R2,A ;OTHERWISE SAVE SUM IN R2
12 00010A 23F7 MOV A,#0F7H ; AND CALL EXIT SVC
13 00010C 00 NOP ; TO END PROGRAM EXECUTION
14 00010D A3 MOVP A,@A
15 00010E 00 NOP
16 ;SRB POINTER
17 0040 ORG 40H ;STORE SRB POINTER AT ADDRESS 40
18 000040 0042 BYTE 00,42H ;POINT TO SRB FOR EXIT SVC
19 ;SRB FOR EXIT SVC
20 000042 1A BYTE 1AH ;1AH = FUNCTION CODE FOR EXIT SVC
21 ;TABLE OF NUMBERS TO BE ADDED
22 TSIZE EQU 5 ;TABLE SIZE = 5
23 SECTION INTERNAL ;SECTION FOR INTERNAL DATA MEMORY
24 ORG 32H ;SET UP TABLE AT ADDRESS 32
25 TABLE BLOCK TSIZE
26 LIST DBG
27 END START
== =====
| | | source code comments
| | |
| | +-- object code
| | 
| +----- address
+----- source code line number

```

3967-12

Fig. 7H-10. Demonstration program.

```

(A)  %276C33100B932BF05276119EF05AA23F700A300
      %0E62B24000421A
      %393464DEM0010350514LOOP310515START310015TABLE23225TSIZE15
      %233318INTERNAL04400023715TABLE44032
      %098153100

```

(B) FIRST DATA BLOCK: object code for addresses 100--10E

header	load address	object code
=====		
%276C33100B932BF05276119EF05AA23F700A300		

SECOND DATA BLOCK: object code for addresses 40--42

header	load address	object code
=====		
%0E62B24000421A		

FIRST SYMBOL BLOCK

header	section name	section definition field	symbol definition fields
=====			
%393464DEM0010350514LOOP310515START310015TABLE23225TSIZE15			

SECOND SYMBOL BLOCK

header	section name	section definition field	symbol definition field
=====			
%233318INTERNAL04400023715TABLE44032			

TERMINATION BLOCK

header	transfer address
=====	
%098153100	

3967-13

Fig. 7H-11. Demonstration program: Extended Tekhex format.

Figure 7H-11A shows an Extended Tekhex load module that contains the object code and program symbols for the demonstration program. Figure 7H-11B labels the different fields in the message blocks. If you have a host computer

other than an 8560, you can create this load module and
download it to your 8540 or 8550.

EXAMINE THE DEMONSTRATION RUN PROGRAM

The demonstration run program adds five numbers from a table stored in 8048 internal data memory locations 32--36 and leaves the sum in register R2. (You will place values in the table later in this demonstration.) The 8085A emulator demonstration run in the Learning Guide section of your System Users Manual contains a flowchart that illustrates the steps of the program.

The source code contains two kinds of statements: assembler directives (such as ORG and BYTE) and 8048 assembly language instructions. The assembler directives are microprocessor-independent and are explained in the 8085A emulator demonstration run. The 8048 assembly language instructions are discussed in the following paragraphs.

Set Table Pointer. The MOV R1,#TABLE instruction loads the lower byte of the address of the table (4032) from internal data memory into register R1. As a result, R1 points to the first element of the table. The label START is used by the END directive to specify that the MOV R1,#TABLE instruction is the first to be executed.

Set Pass Counter. Register R7 is used as the pass counter. The MOV R7,#TSIZE instruction moves the value of TSIZE into R7. This step sets the number of passes to 5. Each time a number is taken from the table and added into the accumulator, R7 is decremented.

Clear Accumulator. The CLR A instruction zeros the accumulator so that you can start adding numbers from the table.

Add Byte From Table. The ADD A,@R1 instruction adds the byte addressed by R1 into the accumulator. The label LOOP represents the address of this instruction; this label is used by the DJNZ instruction.

Point to Next Byte. The INC R1 instruction increments R1; this register then points to the next byte in the table. For example, R1 is initialized to contain 32. After the INC R1 instruction is first executed, R1 will contain 33, the address of the second element of the table.

Decrement Pass Counter and Loop If Not Five Passes Yet. The DJNZ R7,LOOP instruction decrements R7, the pass counter. In this program, R7 is decremented each time a number is added to the accumulator. If R7 does not contain zero after being decremented, the program jumps back to the LOOP label. If R7 contains zero, the program proceeds to the next instruction, MOV R2,A.

Save Sum in R2. After all five numbers have been added together, the `MOV R2,A` instruction moves the sum from the accumulator into register R2. The sum must be saved in R2 because the accumulator is used by the service call that follows.

Exit. The last four instructions of the program are:

```
MOV A,#0F7H
NOP
MOVP A,@A
NOP
```

These instructions constitute a service call (SVC) that causes an exit from the program. For more information on SVCs, refer to the Service Calls discussion in this Emulator Specifics section.

ASSEMBLE AND LOAD THE DEMONSTRATION PROGRAM

Now it's time to create the program so you can run it on your emulator. One of the following discussions describes the set of steps that is appropriate for your hardware configuration:

- For 8550 users -- Case 1: Assemble and Load on the 8550
- For 8540/8560 users -- Case 2: Assemble on the 8560; Download to the 8540
- For 8540 users with a host computer other than the 8560 -- Case 3: Download from Your Host to the 8540
- For other hardware configurations -- Case 4: Patch the Program into Memory

Go ahead and work through the discussion that's appropriate for you. Once you've put the program into 8550/8540 program memory, turn to the heading "Run the Demonstration Program," later in this section.

CASE 1: ASSEMBLE AND LOAD ON THE 8550

This discussion shows you how to copy the demonstration program from your 8048/8021/8041A/8022 emulator software installation disk, assemble the program, and load it into 8550 program memory.

Start Up and Log On

Turn on your 8550 system. (For start-up instructions, refer to the paragraph "Start Up the 8550 and Its Peripherals" in the Learning Guide section of your System Users Manual.) Place your system disk in disk drive 0 and shut the drive 0 door. When you see the > prompt on your system terminal, place your installation disk in disk drive 1 and shut the drive 1 door.

Use the DAT command to set the date and time. For example, if it is 2:35 pm on April 1, 1983, type:

> DAT 01-APR-83/2:35 pm <CR>

Use the SEL command to tell DOS/50 to use the assembler and emulator software designed for the 8048 family:

> SEL 8048 <CR>

The SEL command automatically sets the emulation mode to 0.

Copy the Demonstration Run Program from the Installation Disk

Enter the following command lines to create an empty directory called DEMO on your system disk and make DEMO the current directory. The BR command creates a brief name, ROOT, to mark the old current directory. At the end of this demonstration, you will return to this ROOT directory and delete the DEMO directory and its contents.

> BR ROOT /USR <CR>

> CREATE DEMO <CR>

> USER DEMO <CR>

Now use the COP command to copy all the files in the DEMO2 directory on the installation disk to the DEMO directory you just created:

> COP /VOL/EMU.8048/DEMO2/* * <CR>

Remove your installation disk from drive 1 and put it away.

Now list the files you have just copied to the current directory:

> L <CR>

Filename

ASM

LOAD

Files used	96
Free files	160
Free blocks	738
Bad blocks	0

The file named ASM contains the assembly language source code for the demonstration program, and the file named LOAD contains the executable object code. This copy of LOAD will be used in the demonstration only if you do not have an 8048/8021/8041A/8022 assembler (and thus cannot create your own object file and load file from the source file.)

Examine the Demonstration Program

Enter the following command line to display the source file ASM on the system terminal:

```
> CON ASM <CR>
;8048 DEMONSTRATION RUN PROGRAM
SECTION DEMO
ORG 100H ;START PROGRAM CODE AT ADDRESS 100
START MOV R1,#TABLE ;SET TABLE POINTER
MOV R7,#TSIZE ;SET PASS COUNTER
CLR A ;CLEAR ACCUMULATOR
LOOP ADD A,@R1 ;ADD BYTE FROM TABLE
INC R1 ;POINT TO NEXT BYTE
DJNZ R7,LOOP ;DECREMENT PASS COUNTER AND
; LOOP IF NOT FIVE PASSES YET
MOV R2,A ;OTHERWISE SAVE SUM IN R2
MOV A,#0F7H ; AND CALL EXIT SVC
NOP ; TO END PROGRAM EXECUTION
MOVP A,@A
NOP
;SRB POINTER
ORG 40H ;STORE SRB POINTER AT ADDRESS 40
BYTE 00,42H ;POINT TO SRB FOR EXIT SVC
;SRB FOR EXIT SVC
BYTE 1AH ;1AH = FUNCTION CODE FOR EXIT SVC
;TABLE OF NUMBERS TO BE ADDED
TSIZE EQU 5 ;TABLE SIZE = 5
SECTION INTERNAL
ORG 32H ;SET UP TABLE AT ADDRESS 32
TABLE BLOCK TSIZE
LIST DBG
END START ;END OF SOURCE CODE
```


Assemble the Source Code

If you do not have a 8048/8021/8041A/8022 assembler on your system disk, you cannot perform this step, so skip the next four commands (ASM, COP, LINK, and L.)

The ASM (assemble) command translates assembly language (source code) into binary machine language (object code). The ASM command also creates an assembler listing that can be used to correlate the object code with the source code. Enter the following command line to assemble the source code in the file ASM and create the listing and object files ASML and OBJ:

```
> ASM OBJ ASML ASM <CR>
      |      |      |
      |      |      +--- source file
      |      +----- assembler listing file
      +----- object file
Tektronix      8048 ASM Vx.x
**** Pass 2
    26 Source Lines  26 Assembled Lines  xxxxx Bytes Available
      >>> No assembly errors detected <<<
```

Enter the following command to copy the assembler listing onto the line printer. (First, make sure the printer is turned on and properly connected.)

```
> COP ASML LPT <CR>
```

The different fields of your source listing are presented in Fig. 7H-10, earlier in this demonstration. For a detailed explanation of assembler listings, consult your Assembler Users Manual.

Link the Object Code. The linker creates an executable load file from one or more object files. Enter the LINK command to invoke the linker:

```
> LINK <CR>
8550 LINKER Vx.x
*
```

Now enter the following linker commands to create a load file called LOAD from your object file, OBJ:

```
*LINK OBJ <CR>
*LOAD LOAD <CR>
*LOCATE INTERNAL,BASE(4000H) <CR>
*DEBUG <CR>
*END <CR>
```

The linker commands LINK and LO specify the object file and load file, respectively. The LOCATE command establishes the base address for section INTERNAL. The DEBUG command causes the linker to pass the program symbols from the object file along to the load file, for use in program debugging. After you enter the END command, the linker executes the commands you have entered, and the following information is displayed:

TRUNCATION ERROR AT 0101 IN MODULE *NONAME*
FILE OBJ

2 ERRORS NO UNDEFINED SYMBOLS
1 MODULE 2 SECTIONS
TRANSFER ADDRESS IS 0100

NOTE

Because the instruction MOV R1,#TABLE is a two-byte instruction, the linker truncates 4032 to 32, and displays an error message. Ignore this truncation error message.

The files generated by the ASM and LINK commands should now be on your disk. Enter the following command to list the files in your current directory:

> L <CR>

FILENAME

ASM
LOAD
OBJ
ASML

Files used	126
Free files	130
Free blocks	811
Bad blocks	0

Notice that there are now four files listed in your directory. OBJ and ASML were created by the assembler, and LOAD was created by the linker.

Load the Program into Memory

Now it's time to load the object code from the load file LOAD into program memory.

Zero Out Memory. Before you load any code, use the F (Fill) command to fill 8550/8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory from address 40 through address 11F with zeros:

```
> F 40 11F 00 <CR>
```

Check That Memory Was Filled with Zeros. Check the contents of memory with the D (Dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

```
> D 40 11F <CR>
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Load the Object Code into Memory. Enter the following command line to load the object code for the demonstration program into program memory:

```
> LO <LOAD <CR>
```

```
====
```

```
|
```

```
load file
```

Load the Program Symbols. Recall that the source code for the demonstration program contained the directive LIST DBG. Because of this directive, the object file contains a list of the symbols that appeared in the source code, and the values associated with those symbols. Because you included the DEBUG command when you invoked the linker, those symbols were passed to the load file. Use the SYMLO command to load those symbols into the symbol table in 8550 system memory.

> SYMLO -S <LOAD <CR>

The -S option means that both addresses and scalars are loaded. If you omit the -S, only addresses are loaded. (A scalar is a number that is not an address --- for example, TSIZE, the length of the table.)

Later in this demonstration, whenever you use a symbol in a command line, the operating system refers to the symbol table to find the value that the symbol represents.

You've assembled and linked the demonstration program and loaded it into memory. Now skip ahead to the heading "Run the Demonstration Program."

CASE 2: ASSEMBLE ON THE 8560; DOWNLOAD TO THE 8540

This discussion shows you how to create the demonstration program source code and assemble it on the 8560, then download it to 8540 (or 8550) program memory. If your 8560 does not have an 8048/8021/8041A/8022 assembler, you cannot complete this part of the demonstration, so skip ahead to the heading "Case 4: Patch the Program into Memory" for instructions.

Start Up and Log In

Start up your 8540, make sure it's in TERM mode, and log in to the 8560 operating system, TNIX. See your 8560 System Users Manual for details.

Since you're logged in to TNIX, your system prompt is "\$". (Later in the demonstration, we'll show the system prompt as ">", in deference to people using 8540s and 8550s in LOCAL mode.) Every command you enter is processed by TNIX. If you enter an OS/40 command, TNIX passes it to the 8540.

Enter the following command to select the 8048 assembler on the 8560 and the 8048 emulator on the 8540:

```
$ sel 8048 <CR>
$ uP=8048; export uP <CR>
```

The sel command automatically sets the emulation mode to 0.

Create the Demonstration Program

Enter the following TNIX command lines to create an empty directory called demo and make demo the working directory. You'll create your source file and related files in this demo directory.

```
$ mkdir demo <CR>
$ cd demo <CR>
```

Now use the TNIX editor, ed, to create the demonstration program source file. The following command line invokes the editor and specifies that you want to create a file called asm:

```
$ ed asm <CR>
?asm
```

The editor responds " ?asm" to remind you that asm does not already exist. Notice that the editor does not give a prompt to let you know it's ready for input.

Enter the Text. Now enter the editor command a (append text) and type in the program. Use the BACKSPACE key to erase typing mistakes.

```

a <CR>
      column  column  column
      8      16      26
      |      |      |
      v      v      v
;8048 DEMONSTRATION RUN PROGRAM <CR>
SECTION DEMO <CR>
ORG 100H ;START PROGRAM CODE AT ADDRESS 100 <CR>
START MOV R1,#TABLE ;SET TABLE POINTER <CR>
MOV R7,#TSIZE ;SET PASS COUNTER <CR>
CLR A ;CLEAR ACCUMULATOR <CR>
LOOP ADD A,@R1 ;ADD BYTE FROM TABLE <CR>
INC R1 ;POINT TO NEXT BYTE <CR>
DJNZ R7,LOOP ;DECREMENT PASS COUNTER AND <CR>
; LOOP IF NOT FIVE PASSES YET <CR>
MOV R2,A ;OTHERWISE SAVE SUM IN R2 <CR>
MOV A,#0F7H ; AND CALL EXIT SVC <CR>
NOP ; TO END PROGRAM EXECUTION <CR>
MOVP A,@A <CR>
NOP <CR>
;SRB POINTER <CR>
ORG 40H ;STORE SRB POINTER AT ADDRESS 40 <CR>
BYTE 00,42H ;POINT TO SRB FOR EXIT SVC <CR>
;SRB FOR EXIT SVC <CR>
BYTE 1AH ;1AH = FUNCTION CODE FOR EXIT SVC <CR>
;TABLE OF NUMBERS TO BE ADDED <CR>
TSIZE EQU 5 ;TABLE SIZE = 5 <CR>
SECTION INTERNAL <CR>
ORG 32H ;SET UP TABLE AT ADDRESS 32 <CR>
TABLE BLOCK TSIZE <CR>
LIST DBG <CR>
END START ;END OF SOURCE CODE <CR>
. <CR>

```

At the end of your text, enter a period on a line by itself. The editor will go back to accepting commands.

Check for Errors. Enter the following editor command to display the text you have entered. Check for typing mistakes.

```

1,$p <CR>
| |
| | +-- print command: displays the lines
| |   in the designated range
| |
| | +--- designates last line in file
|
+----- designates first line in file

```

If you made any mistakes, go ahead and fix them. In case you're not familiar with ed, Table 7H-10 lists the commands you need in order to add, delete, or replace any line. For more information on ed, refer to your 8560 System Users Manual.

Table 7H-10
Basic 8560 Editing Commands

Command	Function
mm,nnp <CR>	Displays lines mm through nn
nn <CR>	Makes line nn the current line
d <CR>	Deletes the current line
a <CR> <line(s) of text> . <CR>	Adds text after the current line
c <CR> <line(s) of text> . <CR>	Replaces the current line with the text you type in

Once your text is correct, enter the w command to write the text to the source file, asm:

```
w <CR>
778
```

The editor responds with the number of characters it wrote to the file.

Finally, enter the q command to quit the editor and return to TNIX:

```
q <CR>
$ <--- TNIX prompt
```

Assemble the Source Code.

The TNIX asm (assemble) command translates assembly language (source code) into binary machine language (object code). The asm command also creates an assembler listing which can be used to correlate the object code with the source code. Enter the following command line to assemble the source code in the file asm and create the listing and object files asml and obj:

```
$ asm obj asml asm <CR>
      |      |      |
      |      |      +-- source file
      |      +----- assembler listing file
      +----- object file
```

```
ASM MCS 8048 Vxx.xx-xx Copyright 198x Tektronix, Inc.
*****Pass 2
```

```
27 Lines Read
27 Lines Processed
0 Errors
```

Enter the following command to print the assembler listing on the 8560's line printer:

\$ lp1r asml <CR>

Check page 1 of your listing. Did the assembler issue any error messages? There should be none. If your source code contains errors, take the following steps:

1. Refer to your Assembler Users Manual to find out what the error messages mean.
2. Enter the command ed asm to get back into the editor and fix the mistakes in your source code. Exit the editor with the w and q commands, as before.
3. Enter the command asm obj asml asm to re-assemble your source code.

Link the Object Code

The linker creates an executable load file from one or more object files. Enter the following command to create a load file called load from your object file, obj. Be sure to capitalize all parameters as shown.

\$ link -d -O obj -o load -m prog=0-120h -m int=4000-4037h -L sec=DEMO range prog -L sec=INTERNAL range int <CR>

NOTE

To simplify your task, you may want to create a linker command file to hold the command options. To do so, invoke ed, and place each command option on a separate line. Then, to execute the command file, you would enter:
\$ link -c commandfilename.

The -d option causes the linker to pass the program symbols from the object file to the load file, for use in program debugging. The -m option defines the memory map. In this case, the areas mapped are in program memory and internal data memory. The -L option locates the two sections DEMO and INTERNAL.

The linker responds with the following truncation error message. The truncation errors are caused by the MOV R1,#TABLE and DJNZ R7,LOOP instructions. These instructions each expect an 8-bit operand. You may ignore these messages.

```
link:115 (E) Truncation error at      101
link:115 (E) Truncation error at      108
```


The files generated by the asm and link commands should now be in your working directory, demo. Enter the following command to list the files in your working directory:

```
$ ls <CR>
asm
asml
load
obj
```

Notice that there are now four files listed in your directory. obj and asml were created by the assembler, and load was created by the linker.

Download the Program to the 8540

Now it's time to download the object code produced by the 8560's linker into 8540 program memory.

Zero Out Memory. Before you download any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory from address 40 through address 11F with zeros:

```
$ f 40 11f 00 <CR>
```

Check That Memory Was Filled with Zeros. Check the contents of memory with the OS/40 D (Dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

```
$ d 40 11f <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Download the Object Code. Enter the following command line to download the object code from the 8560 file load to 8540 program memory:

```
$ lo <load <CR>
    ====
    |
load file
```

Download the Program Symbols. Recall that the source code for the demonstration program contains the directive LIST DBG. Because of this directive, the object file contains a list of the symbols that appear in the source code, and the values associated with those symbols. Because you included the -d option in the link command line, those symbols were passed to the load file. Use the OS/40 SYMLO command to download those symbols into the symbol table in 8540 system memory.

```
$ symlo -s <load <CR>
```

The -S option means that both addresses and scalars are downloaded. If you omit the -S, only addresses are downloaded. (A scalar is a number that is not an address --- for example, TSIZE, the length of the table.)

Later in this demonstration, whenever you use a symbol in an OS/40 command line, OS/40 refers to the symbol table to find the value that the symbol stands for.

You've assembled and linked the demonstration program and downloaded it into memory. Now skip ahead to the heading "Run the Demonstration Program."

CASE 3: DOWNLOAD FROM YOUR HOST TO THE 8540

This discussion gives some general instructions for downloading the demonstration program from an unspecified host computer to 8540 (or 8550) program memory. If your 8540 is not equipped with the optional COM Interface Package, you cannot complete this part of the demonstration, so skip ahead to the heading "Case 4: Patch the Program into Memory" for instructions. COM Interface software is standard on the 8550.

Since we don't know what host computer you're using, we can only provide a general outline for creating the demonstration program and downloading it to the 8540. Once you have determined the command sequence that is appropriate for your host, record this information in the space provided in Fig. 7H-12.

Create the Extended Tekhex Load Module

In order for the object code to be downloaded to the 8540, it must be in Extended Tekhex format, as shown in Fig. 7H-11, earlier in this demonstration. You can create the load module in one of two ways:

1. Use your host computer's text editor, and key the load module in by hand.
2. Use your host computer's 8048 assembler:
 - a. Translate the demonstration program into the language of your host's 8048 assembler.
 - b. Create and assemble the source file.
 - c. Link the object code, if necessary.
 - d. Translate the object code produced by the assembler or linker into Extended Tekhex format. The Intersystem Communication section of your System Users Manual provides a general algorithm for conversion to Extended Tekhex format.

Prepare the 8540

Start up your 8540 and enter the following command to select the 8048 emulator:

```
> SEL 8048 <CR>
```

The SEL command automatically sets the emulation mode to 0.

Create the Extended Tekhex Load ModulePrepare the 8540

(Start up the 8540.)

```
> SEL 8048 <CR>  
> F 40 11F 00 <CR>  
> D 40 11F <CR>
```

Establish CommunicationDownload the Load ModuleTerminate Communication

3967-14

Fig. 7H-12 Host computer commands for preparing demonstration program

Zero Out Memory. Before you download any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. (Zeroing out memory has no effect on how the program is loaded.) Enter the following command line to fill memory from address 40 through address 11F with zeros:

```
> F 40 11F 00 <CR>
```

Check That Memory Was Filled with Zeros. Check the contents of memory with the OS/40 D (Dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

```
> D 40 11F <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000100 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Download the Load Module to the 8540

Be sure that your 8540 and your host computer are connected via an RS-232-C-compatible communications link. Then perform the following steps to download the Tekhex load module to 8540 program memory. (Refer to the Intersystem Communication section of your System Users Manual to determine the commands and parameters that are appropriate for your host computer.)

- a. Enter the 8540 COM command to establish communication. (The parameters of the COM command are host-specific.) Log on to your host and execute any necessary host initialization commands.
- b. Enter the command line that downloads the Tekhex load module to the 8540. This command line consists of a host computer command that performs the download, followed by a null character (CTRL-@ on most terminals) and a carriage return. COM places the object code in 8540 program memory, and puts the program symbols into the symbol table in 8540 system memory.
- c. Log off from your host, and then terminate COM command execution by entering the null character, then pressing the ESC key.

Once you've downloaded the program to the 8540, skip ahead to the heading "Run the Demonstration Program."

CASE 4: PATCH THE PROGRAM INTO MEMORY

This discussion shows you how to patch the demonstration program into 8540 (or 8550) program memory using the P command, and then add the program symbols into the symbol table using the ADDS command.

Ordinarily, you would load the object code and symbols from a binary or hexadecimal load file, as illustrated for Cases 1, 2, and 3. The procedure presented here is not normally used for preparing a program for execution. Use this procedure only if you have no standard means for preparing the program, but would still like to try out your emulator.

Start Up the 8540

Start up your 8540 and enter the following command to select the 8048 emulator:

```
> SEL 8048 <CR>
```

The SEL command automatically sets the emulation mode to 0.

Zero Out Memory

Before you patch in any code, use the OS/40 F (Fill) command to fill 8540 program memory with zeros. Later, when you examine memory, the zeros make it easy to identify the beginning and end of your code. Enter the following command line to fill memory from address 40 through address 11F with zeros:

```
> F 40 11F 00 <CR>
```

Check That Memory Was Filled with Zeros. Check the contents of memory with the OS/40 D (Dump) command. The D command's display shows the data in hexadecimal format, and also shows the corresponding ASCII characters. Display the contents of memory addresses 40--11F with the following command line:

```
> D 40 11F <CR>
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0000F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Patch the Object Code into Memory

The OS/40 P (Patch) command stores a sequence of bytes into memory, replacing the previous memory contents. Enter the following command to store the object code for the first three instructions in the program (MOV R1, MOV R7, and CLR A) starting at location 100:

```
> P 100 B932 BF05 27 <CR>
    ===  =====  ==
    |      |      |      |
    |      |      |      | CLR A
    |      |      |      |
    |      |      |      | MOV R7,#TSIZE
    |      |      |      |
    |      |      |      | MOV R1,#TABLE
    |      |      |      |
patch address
```

Now patch in the next four instructions (ADD, INC, DJNZ, and MOV R2)...

```
> P 105 61 19 EF05 AA <CR>
```

... and now the last four instructions (MOV A, NOP, MOVP A, and NOP).

```
> P 10A 23F7 00 A3 00 <CR>
```

Finally, patch in the Exit SVC information at address 40:

```
> P 40 00421A <CR>
```

You'll check the contents of memory later in this demonstration.

Put Symbols into the Symbol Table

Later in this demonstration, you will use symbols from the demonstration program (START, LOOP, TSIZE, and TABLE) when communicating with OS/40. Whenever you use a symbol in a command line, OS/40 consults a symbol table in 8540 system memory to find the values that the symbol stands for. Enter the following command line to add the program symbols to the symbol table, along with their values:

```
> ADDS START=100 LOOP=106 -S TSIZE=5 TABLE=4032 <CR>
```

The ADDS command cannot provide all the symbol-related information that is provided by the SYML0 command (as in Cases 1 and 2) or the COM command (as in Case 3). Because this information is missing, some of the displays you produce later in this demonstration will not match the symbolic displays shown in this manual. For more information on the ADDS command, refer to the Command Dictionary of your System Users Manual.

You've patched the demonstration program into program memory and placed the program symbols in the symbol table. Now it's time to run the program.

 RUN THE DEMONSTRATION PROGRAM

From now until the end of the demonstration, the commands you are to enter are shown in lowercase. If you are not logged in to an 8560, you may enter commands in either lowercase or uppercase. If you are using an 8560, you must enter the name of every command in lowercase (and your system prompt is "\$", not ">").

Now that you've loaded the program into memory, you need to:

- verify that the program was loaded correctly; and
- put values into the table in memory, for the program to add.

Check Memory Contents Again. Before you loaded the program, you filled memory locations 40--11F with zeros. Look at the same memory area again with the following command line:

```
> d 40 11f <CR>
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0040 00 42 1A 00 00 00 00 00 00 00 00 00 00 00 00 00 .B.....
0050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0100 B9 32 BF 05 27 61 19 EF 05 AA 23 F7 00 A3 00 00 .2..'a....#.....
0110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

The object code is loaded in two different blocks:

- The 8048 machine instructions are loaded at address 100 (specified by the first ORG directive in the source code).
- The information for the Exit SVC is loaded at address 40 (specified by the second ORG directive).

The contents of the table at address 32 are still undefined, but you'll put some values into the table in just a few minutes.

Turn On Symbolic Display. Enter the following command to tell the system to modify its displays by replacing hexadecimal numbers with symbols from your program, where appropriate.

```
> synd on <CR>
```

Disassemble the Object Code. The DI (Disassemble) command displays memory contents both in hexadecimal notation and in assembly language mnemonics. You can use the DI command to verify that the object code in memory corresponds to your source code. Enter the following command to disassemble the area of memory occupied by the executable part of your program:

> di 100 10e <CR>

LOC	INST	MNEM	OPER
SECTION (DEMO)			
START	B932	MOV	R1,#32
+000102	BF05	MOV	R7,#05
+000104	27	CLR	A
LOOP	61	ADD	A,@R1
+000106	19	INC	R1
+000107	EF05	DJNZ	R7,#05 0105
+000109	AA	MOV	R2,A
+00010A	23F7	MOV	A,#F7
+00010C	00	NOP	
+00010D	A3	MOVP	A,@A
+00010E	00	NOP	
+00010E	12	NOP	

Compare the DI display with the assembler listing you generated earlier, or refer back to Fig. 7H-10.

The line "SECTION (DEMO)" in the DI display indicates that the object code being disassembled comes from the program section called DEMO. In fact, the entire memory area used by your program (location 0 through the end of the program --- location 10E) belongs to section DEMO. This section was declared by the SECTION directive in the source code. (If you used the ADDS command to create your symbols, as in Case 4, the section name shown in the DI display is NO.SECTION.)

The LOC (location) column of the DI display contains information that enables you to correlate the display with your assembler listing. The symbols START and LOOP in the DI display correspond to the labels START and LOOP in the source code. For those lines of the display where the location does not correspond to a label in the symbol table, DI substitutes the address of the instruction relative to the beginning of the section, as shown in the address field of your assembler listing. If you haven't loaded the pertinent symbols and related information into the symbol table (using a command such as SYML0), the DI command supplies absolute (actual) addresses in the LOC column. (Since section DEMO begins at address 0, the relative address, or offset, is the same as the absolute address in this display. This offset feature is much more useful for sections that don't start at address 0.)

Now you've seen that your system can use the symbol table to translate numbers into symbols to make a display easier to read. Your system can also translate a symbol in a command line into an address. For example, since your system knows that the symbol START is equivalent to the address 100, you could have entered the DI command in any of the following ways:

```
di 100 10E
di START 10E
di start start+0e
di 100 START+0E
```

Notice that a symbol can be entered in either lowercase or uppercase.

The feature that enables DOS/50 and OS/40 to correlate symbols from your program with the numbers they represent is termed symbolic debug.

Put Values into the Table in Memory. The demonstration program sums five numbers from a table in memory. Use the P (Patch) command to store the numbers 1, 2, 3, 4, and 5 in the table. Do you remember what the address of the table is? It doesn't matter, as long as you remember that the symbol TABLE represents that address.

```
> p table 0102030405 <CR>
      =====
      |      |
address of  string of bytes to be stored
table: 4032 at addresses 4032--4036
```

Check the Contents of the Table. Use the D command to display the contents of the table. (When you don't specify an upper boundary for the area to be dumped, the D command dumps 16 bytes.)

```
+----- lower address: 4032
|
| +--- upper address: omitted
| | (defaults to lower address + 0F)
| |
=====
> d table <CR>
    2 3 4 5 6 7 8 9 A B C D E F 0 1
004032 01 02 03 04 05 27 EB 8F C3 3C EB B6 9D 2B 00 42 ...'...<...+...B
```

Notice that bytes 4032--4036 (the table) contain the values you patched in. Bytes 4037--4041 contain random data left over from previous system operations.

The following command dumps only the contents of the table:

```
> d table table+tsize-1 <CR>
    2 3 4 5 6 7 8 9 A B C D E F 0 1
004032 01 02 03 04 05
.....
```

Start Program Execution

Enter the G (Go) command to start program execution at location 100, the transfer address specified by the END directive in the source code. (If you followed "Case 4: Patch the Program into Memory," you must enter G START instead.)

> g <CR>

				accumulator		table pointer		sum of table		pass counter								
				v		v		v		v								
LOC	INST	MNEM	OPERAND	EADD	A	PSW	FLAGS	TR	RB	R0	R1	R2	R3	R4	R5	R6	R7	
010D	A3	MOVP	A,@A		00	08	00	00	00	0	19	37	0F	A6	FD	00	BE	00
010D	<BREAK		TRACE>															

The program executes, and when the Exit SVC occurs, the program breaks (stops), and the contents of the emulator registers are displayed. The Exit SVC uses the accumulator, so the sum of the numbers in the table is transferred into R2: 1+2+3+4+5=0F.

MONITOR PROGRAM EXECUTION

You have assembled, loaded, and executed the demonstration program. The rest of this demonstration shows you some commands for monitoring program execution. You can watch the changes in the emulator's registers and observe the effect of each instruction as the program proceeds.

Trace All Instructions. The TRA (TRAcE) command lets you observe the changes in the 8048 registers as the program proceeds. When you enter a TRA command and then start execution with the G command, display lines are sent to the system terminal. As each instruction executes, the display line shows the instruction (as in the DISassemble display) and the contents of the registers after that instruction has executed. Enter the following command line to trace all of the program's instructions:

> tra all <CR>

Enter the command G START (or G 100) to resume program execution at the beginning of the program:

> g start <CR>

As the program executes, the following trace is displayed. Remember that you can type CTRL-S to suspend the display and CTRL-Q to resume the display.

LOC	INST	MNEM	OPERAND	EADD	A	PSW	FLAGS	TR	RB	R0	R1	R2	R3	R4	R5	R6	R7
SECTION (DEMO)																	
START	B932	MOV	R1,#32H		07	08	00	00	00	0	19	32	07	A6	FD	00	BE 00
+000102	BF05	MOV	R7,#05H		07	08	00	00	00	0	19	32	07	A6	FD	00	BE 05
+000104	27	CLR	A		00	08	00	00	00	0	19	32	07	A6	FD	00	BE 05
LOOP	61	ADD	A,@R1		01	08	00	00	00	0	19	32	07	A6	FD	00	BE 05
+000106	19	INC	R1		01	08	00	00	00	0	19	33	07	A6	FD	00	BE 05
+000107	EF05	DJNZ	R7,#05H	0105	01	08	00	00	00	0	19	33	07	A6	FD	00	BE 04
LOOP	61	ADD	A,@R1		03	08	00	00	00	0	19	33	07	A6	FD	00	BE 04
+000106	19	INC	R1		03	08	00	00	00	0	19	34	07	A6	FD	00	BE 04
+000107	EF05	DJNZ	R7,#05H	0105	03	08	00	00	00	0	19	34	07	A6	FD	00	BE 03
LOOP	61	ADD	A,@R1		06	08	00	00	00	0	19	34	07	A6	FD	00	BE 03
+000106	19	INC	R1		06	08	00	00	00	0	19	35	07	A6	FD	00	BE 03
+000107	EF05	DJNZ	R7,#05H	0105	06	08	00	00	00	0	19	35	07	A6	FD	00	BE 02
LOOP	61	ADD	A,@R1		0A	08	00	00	00	0	19	35	07	A6	FD	00	BE 02
+000106	19	INC	R1		0A	08	00	00	00	0	19	36	07	A6	FD	00	BE 02
+000107	EF05	DJNZ	R7,#05H	0105	0A	08	00	00	00	0	19	36	07	A6	FD	00	BE 01
LOOP	61	ADD	A,@R1		0F	08	00	00	00	0	19	36	07	A6	FD	00	BE 01
+000106	19	INC	R1		0F	08	00	00	00	0	19	37	07	A6	FD	00	BE 01
+000107	EF05	DJNZ	R7,#05H	0105	0F	08	00	00	00	0	19	37	07	A6	FD	00	BE 00
+000109	AA	MOV	R2,A		0F	08	00	00	00	0	19	37	0F	A6	FD	00	BE 00
+00010A	23F7	MOV	A,#F7H		F7	08	00	00	00	0	19	37	0F	A6	FD	00	BE 00
+00010C	00	NOP			F7	08	00	00	00	0	19	37	0F	A6	FD	00	BE 00
+00010D	A3	MOVP	A,@A		00	08	00	00	00	0	19	37	0F	A6	FD	00	BE 00
+00010D	<BREAK		TRACE>														

After the accumulator is cleared, the program begins to store the sum of the numbers being added. The 8048 ADD A,@R1 instruction adds a number from the table into the accumulator. Because the accumulator is also used for the Exit SVC, the program transfers the sum of numbers (1+2+3+4+5=0F) from the accumulator into R2.

Register R7, the pass counter, is set to contain 5 at the beginning of the program. It decreases by one (because of the DJNZ instruction) each time a number is added into the accumulator. The program ends after register R7 reaches zero.

The R1 register, set to contain 32 at the start of the program, increments (because of the INC R1 instruction) each time a number is added to the accumulator. At the end of the program, the R1 register has been incremented five times and contains 37.

Trace to the Line Printer. By adding the parameter >LPT to a command, you can direct that command's output to the line printer instead of to the system terminal. First, verify that your line printer is properly connected and powered up. Then enter the following command to execute the program with trace output directed to the line printer:

```
> g start >LPT <CR>
```

NOTE

If you're operating in TERM mode with an 8560, use one of the following commands in place of the command shown:

- g start | lp1r sends the display to the 8560 line printer.
- g start \>LPT sends the display to the line printer on the 8540 or 8550.

Trace Jump Instructions Only. Another way to monitor the program's execution is to look only at the jump instructions. By tracing the jump instructions, you can still observe the changes in the registers, but you save time and space by not tracing the instructions within the loop. Enter the following command line to trace only the jump instructions when the loop is being executed:

```
> tra jmp loop 108   <CR>
      |   |
      |   +--- upper address } Within this range,
      |                           } only jump instructions
      +----- lower address } are traced.
                        (105)
```

Again, enter the G command line to start program execution. The following trace is displayed:

```
> g start   <CR>
```

LOC	INST	MNEM	OPERAND	EADD	A	PSW	FLAGS	TR	RB	R0	R1	R2	R3	R4	R5	R6	R7		
SECTION (DEMO)																			
START	B932	MOV	R1,#32H			07	08	00	00	00	0	19	32	07	A6	FD	00	BE	00
+000103	61	ADD	A,@R1			01	08	00	00	00	0	19	32	07	A6	FD	00	BE	05
+000105	19	INC	R1			01	08	00	00	00	0	19	33	07	A6	FD	00	BE	05
+000107	EF05	DJNZ	R7,#05H	0105		01	08	00	00	00	0	19	33	07	A6	FD	00	BE	04
+000107	EF05	DJNZ	R7,#05H	0105		03	08	00	00	00	0	19	34	07	A6	FD	00	BE	03
+000107	EF05	DJNZ	R7,#05H	0105		06	08	00	00	00	0	19	35	07	A6	FD	00	BE	02
+000107	EF05	DJNZ	R7,#05H	0105		0A	08	00	00	00	0	19	36	07	A6	FD	00	BE	01
+000107	EF05	DJNZ	R7,#05H	0105		0F	08	00	00	00	0	19	37	07	A6	FD	00	BE	00
+000109	AA	MOV	R2,A			0F	08	00	00	00	0	19	37	0F	A6	FD	00	BE	00
+000109	<BREAK		TRACE>																

As with the TRA ALL display, observe that register R7 (the pass counter)

is decremented. The R1 register (the table pointer) is incremented, and the accumulator stores the sum of the numbers from the table. With the TRA JMP command in effect, the instructions within the loop are not displayed.

Check the Status of the Trace. The TRA command without any parameters displays the trace conditions that are presently set. Because you can have up to three trace selections in effect at the same time, it is useful to be able to see which trace selections are active. Check your trace status with the following command line:

```
> tra <CR>

TRACE    ALL,000000,00FFFF
TRACE    JMP,LOOP,0108
```

As you've specified, TRA ALL is in effect for addresses 0--104, TRA JMP is in effect for addresses 105--108, and TRA ALL is again in effect for addresses 109--FFFF.

Set a Breakpoint after a Specific Instruction. Now that you've seen how the program adds the numbers together, here's a new task: to add only the third and fourth numbers from the table. To perform this task, you want the pass counter to contain 2, and the table pointer to contain 34 (the address of the third number in the table). You can accomplish these changes without altering the object code in program memory. First, stop program execution after the pass counter and the table pointer have been set. While the program is stopped, enter new values for the pass counter and the table pointer. When execution resumes, the program treats the new values as if they were the original programmed values.

Enter the following command line to trace all of the instructions as the program executes:

```
> tra all <CR>
```

Check the trace status with the following command line:

```
> tra <CR>

TRACE    ALL,0000,FFFF
```

The TRA selections we set earlier are made obsolete by the TRA ALL command just entered.

Now set a breakpoint so that the program stops after the table pointer and pass counter have been set. The following command causes the program to stop after it executes the MOV R7 instruction at address 102:

```
> bk 1 102 <CR>
    = ==
    | |
    | +-- breakpoint address
    |
    +----- breakpoint number
```

Use the G command to start program execution:

[illegible]

The TRA ALL command enabled display of all instructions up to and including the instruction at the breakpoint.

Set New Values in Pass Counter and Table Pointer; Check Results. Now that you've reached the breakpoint, you can change the contents of the registers while execution is stopped. The break display shows that R7 (the pass counter) contains 5, and that the R1 register (the table pointer) points to address 32. Use the S (Set) command to set the number of passes to two and set the table pointer to 34:

```
> s r1=34 r7=02    <CR>
      |      |
      |      +-- contents of register R7
      |
      +----- contents of register R1
```

The `S` command does not produce a display, but you can use the `DS` (Display Status) command to check the values in the registers you changed. `DS` displays the contents of each emulator register and status flag. Check the result of the previous `S` command with the following command line:

> ds <CR>

```

PC      CHIP EPM  A  PSW TR RB          REGISTERS
0104 8048  Y  OF  08 00  0      R0-R7= 19 34 OF A6 FD 00 BE 02
                                     A0-A7= 86 00 2F 40 49 09 7D 00

IIP EI  TI  TC  MB STACK RETURN TF FO F1 AN STF DMA IBF OBF FLG
  N  0  0  0  0  0      08    1F06    0  0  0

```

The DS display shows that the pass counter and the table pointer now contain the new values.

NOTE

The DS display varies slightly for different microcomputers in the 8048 family.

Resume Program Execution. If you enter the G command with no parameters, program execution starts where it left off. Resume program execution after the breakpoint with the following command line:

> g <CR>

LOC	INST	MNEM	OPERAND	EADD	A	PSW	FLAGS	TR	RB	RO	R1	R2	R3	R4	R5	R6	R7
SECTION (DEMO)																	
+000104	27	CLR	A		00	08	00	00	00	0	19	34	0F	A6	FD	00	BE 02
LOOP	61	ADD	A,@R1		01	08	00	00	00	0	19	34	0F	A6	FD	00	BE 02
+000106	19	INC	R1		01	08	00	00	00	0	19	35	0F	A6	FD	00	BE 02
+000107	EF05	DJNZ	R7,#05H	0105	01	08	00	00	00	0	19	35	0F	A6	FD	00	BE 01
LOOP	61	ADD	A,@R1		03	08	00	00	00	0	19	35	0F	A6	FD	00	BE 01
+000106	19	INC	R1		03	08	00	00	00	0	19	36	0F	A6	FD	00	BE 01
+000107	EF05	DJNZ	R7,#05H	0105	03	08	00	00	00	0	19	36	0F	A6	FD	00	BE 00
+000109	AA	MOV	R2,A		0F	08	00	00	00	0	19	36	07	A6	FD	00	BE 00
+00010A	23F7	MOV	A,#F7H		F7	08	00	00	00	0	19	36	F7	A6	FD	00	BE 00
+00010C	00	NOP			F7	08	00	00	00	0	19	36	07	A6	FD	00	BE 00
+00010D	A3	MOVP	A,@A		00	08	00	00	00	0	19	36	07	A6	FD	00	BE 00
+00010D	<BREAK		TRACE>														

Notice that the program performed two passes through the loop, and that the program added the third and fourth numbers in the table ($3 + 4 = 7$).

SUMMARY OF 8048/8021/8041A/8022 EMULATOR DEMONSTRATION RUN

You have assembled, loaded, executed, and monitored the demonstration run program. You have used the following commands:

- SEL -- selects the 8048 assembler and emulator
- ASM -- creates object code from an assembly language program
- LINK -- links object code into a load module
- F -- fills an area of memory with a specified value
- D -- displays memory contents in ASCII and hexadecimal format
- LO -- loads object code into memory
- DI -- translates memory contents into assembly language mnemonics
- P -- patches a string of bytes into memory
- SYMD ON -- turns on symbolic debug feature
- G -- begins or resumes program execution
- TRA -- selects instructions to be traced during program execution
- BK -- sets a breakpoint
- S -- modifies emulator processor registers
- DS -- displays the status of the emulator processor

Delete the Demonstration Run Files

Now that you've finished the demonstration run, you can delete the source file, object file, listing file, and load file. If you're using an 8550, the source and object files are still available to you on the 8048/8021/8041A/8022 emulator installation disk. If you're using an 8560, remember that once you delete the source file (asm), there is no way of recovering it.

Delete 8550 Files. If your files are on the 8550, use the following procedure to delete them. First use the USER command to move from the DEMO directory back into the directory you were in at the start of the demonstration. Recall that you marked that directory with the brief name /ROOT.

```
> USER /ROOT <CR>
```

Now enter the following command to delete the DEMO directory and the files it contains:

```
> DEL /DEMO/* DEMO <CR>
```

```
Delete ASM          ? Y <CR>
Delete LOAD         ? Y <CR>
Delete OBJ          ? Y <CR>
Delete ASML         ? Y <CR>
Delete DEMO         ? Y <CR>
```

Before deleting each file, DOS/50 asks you whether you really want to delete it. You type "Y" for yes.

Delete 8560 Files. If your files are on the 8560, use the following procedure to delete them. Enter the following command to remove all files in the working directory, including the source file:

```
$ rm * <CR>
```

Now move from the demo directory back into the parent directory and remove the demo directory itself:

```
$ cd .. <CR>
$ rmdir demo <CR>
```

Turn Off Your System

For instructions on turning off your 8550 or 8540, refer to the Learning Guide of your System Users Manual.

Resume Program Execution. If you enter the G command with no parameters, program execution starts where it left off. Resume program execution after the breakpoint with the following command line:

> g <CR>

LOC	INST	MNEM	OPERAND	EADD	A	PSW	FLAGS	TR	RB	RO	R1	R2	R3	R4	R5	R6	R7
SECTION (DEMO)																	
+000104	27	CLR	A		00	08	00	00	00	0	19	34	0F	A6	FD	00	BE 02
LOOP	61	ADD	A,@R1		01	08	00	00	00	0	19	34	0F	A6	FD	00	BE 02
+000106	19	INC	R1		01	08	00	00	00	0	19	35	0F	A6	FD	00	BE 02
+000107	EF05	DJNZ	R7,#05H	0105	01	08	00	00	00	0	19	35	0F	A6	FD	00	BE 01
LOOP	61	ADD	A,@R1		03	08	00	00	00	0	19	35	0F	A6	FD	00	BE 01
+000106	19	INC	R1		03	08	00	00	00	0	19	36	0F	A6	FD	00	BE 01
+000107	EF05	DJNZ	R7,#05H	0105	03	08	00	00	00	0	19	36	0F	A6	FD	00	BE 00
+000109	AA	MOV	R2,A		0F	08	00	00	00	0	19	36	07	A6	FD	00	BE 00
+00010A	23F7	MOV	A,#F7H		F7	08	00	00	00	0	19	36	F7	A6	FD	00	BE 00
+00010C	00	NOP			F7	08	00	00	00	0	19	36	07	A6	FD	00	BE 00
+00010D	A3	MOVP	A,@A		00	08	00	00	00	0	19	36	07	A6	FD	00	BE 00
+00010D	<BREAK		TRACE>														

Notice that the program performed two passes through the loop, and that the program added the third and fourth numbers in the table ($3 + 4 = 7$).

SUMMARY OF 8048/8021/8041A/8022 EMULATOR DEMONSTRATION RUN

You have assembled, loaded, executed, and monitored the demonstration run program. You have used the following commands:

- SEL -- selects the 8048 assembler and emulator
- ASM -- creates object code from an assembly language program
- LINK -- links object code into a load module
- F -- fills an area of memory with a specified value
- D -- displays memory contents in ASCII and hexadecimal format
- LO -- loads object code into memory
- DI -- translates memory contents into assembly language mnemonics
- P -- patches a string of bytes into memory
- SYMD ON -- turns on symbolic debug feature
- G -- begins or resumes program execution
- TRA -- selects instructions to be traced during program execution
- BK -- sets a breakpoint
- S -- modifies emulator processor registers
- DS -- displays the status of the emulator processor

Delete the Demonstration Run Files

Now that you've finished the demonstration run, you can delete the source file, object file, listing file, and load file. If you're using an 8550, the source and object files are still available to you on the 8048/8021/8041A/8022 emulator installation disk. If you're using an 8560, remember that once you delete the source file (asm), there is no way of recovering it.

Delete 8550 Files. If your files are on the 8550, use the following procedure to delete them. First use the USER command to move from the DEMO directory back into the directory you were in at the start of the demonstration. Recall that you marked that directory with the brief name /ROOT.

> USER /ROOT <CR>

Now enter the following command to delete the DEMO directory and the files it contains:

> DEL /DEMO/* DEMO <CR>

Delete ASM	? <u>Y</u> <CR>
Delete LOAD	? <u>Y</u> <CR>
Delete OBJ	? <u>Y</u> <CR>
Delete ASML	? <u>Y</u> <CR>
Delete DEMO	? <u>Y</u> <CR>

Before deleting each file, DOS/50 asks you whether you really want to delete it. You type "Y" for yes.

Delete 8560 Files. If your files are on the 8560, use the following procedure to delete them. Enter the following command to remove all files in the working directory, including the source file:

\$ rm * <CR>

Now move from the demo directory back into the parent directory and remove the demo directory itself:

\$ cd .. <CR>
\$ rmdir demo <CR>

Turn Off Your System

For instructions on turning off your 8550 or 8540, refer to the Learning Guide of your System Users Manual.